(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
22 mars 2001 (22.03.2001)

PCT

(10) Numéro de publication internationale
WO 01/20355 A1

(51) Classification internationale des brevets[7]:
G01R 31/3183

(21) Numéro de la demande internationale:
PCT/FR00/02554

(22) Date de dépôt international:
14 septembre 2000 (14.09.2000)

(25) Langue de dépôt: français

(26) Langue de publication: français

(30) Données relatives à la priorité:
99/11534 15 septembre 1999 (15.09.1999) FR

(71) Déposant *(pour tous les États désignés sauf US)*: **CENTRE NATIONAL D'ETUDES SPATIALES** [FR/FR]; 2, place Maurice Quentin, F-75001 Paris (FR).

(72) Inventeurs; et

(75) Inventeurs/Déposants *(pour US seulement)*: **ROLLAND, Guy** [FR/FR]; 3, rue des Lilas, F-31750 Escalquens (FR). **DESPLATS, Romain** [FR/FR]; 5, allée Philippe Ariès, F-31400 Toulouse (FR).

(74) Mandataire: **LE GUEN, Gérard**; Cabinet Lavoix, 2, place d'Estienne d'Orves, F-75441 Paris Cedex 09 (FR).

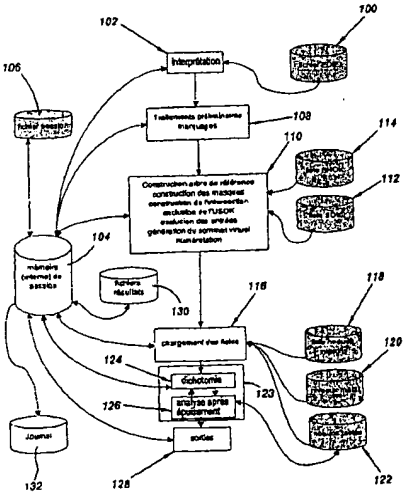
(81) État désigné *(national)*: US.

(84) États désignés *(régional)*: brevet européen (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

[Suite sur la page suivante]

(54) Title: METHOD FOR LOCATING FAULTY ELEMENTS IN AN INTEGRATED CIRCUIT

(54) Titre: PROCEDE DE LOCALISATION D'ELEMENTS DEFECTUEUX DANS UN CIRCUIT INTEGRE



(57) **Abstract:** The invention concerns a method for locating faulty elements in an integrated circuit. The method consists: in modelling the integrated circuit in the form of a tree of nodes and directed arcs; measuring at various nodes of the circuit by applying a testing sequence in the circuit input; recursively determining the nodes to be tested on the basis of the previously performed tests. Each new testing node is such that the number of its ancestors is substantially equal to the number of its descendants.

(57) **Abrégé:** L'invention concerne un procédé de localisation d'éléments défectueux dans un circuit intégré. Le circuit intégré est modélisé sous forme d'un arbre formé de noeuds et d'arcs orientés. Des mesures sont effectuées en différents noeuds du circuit par application d'une séquence de test en entrée du circuit. Les noeuds à tester sont déterminés de manière récursive en fonction du résultat des tests précédemment effectués. Chaque nouveau noeud de test est tel que le nombre de ses ancêtres est sensiblement égal au nombre de ses descendants.

WO 01/20355 A1

Publiée:
— *Avec rapport de recherche internationale.*
— *Avant l'expiration du délai prévu pour la modification des revendications, sera republiée si des modifications sont reçues.*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

Procédé de localisation d'éléments défectueux dans un circuit intégré.

La présente invention concerne un procédé de localisation d'un élément défectueux dans un circuit intégré dont l'agencement théorique est connu, du type comportant une succession d'étapes consistant en :

- la détermination d'un point de mesure du circuit intégré ; et
- le test du point de mesure déterminé par mise en œuvre de :
  - l'application d'une séquence de tests aux entrées du circuit intégré ;
  - la mesure de signaux au point de mesure déterminé du circuit intégré, lors de l'application de la séquence de tests ; et
  - l'évaluation du point de mesure par la comparaison des signaux mesurés avec des signaux théoriques devant être obtenus au point de mesure déterminé pour évaluer si le point de mesure est défaillant ou satisfaisant ; et

dans lequel la position de l'élément défectueux du circuit intégré est déterminée à partir des évaluations effectuées aux différents points de mesure déterminés,

Il est commode, pour la fabrication des circuits intégrés, ou lors de la recherche de défaillances de systèmes industriels, de pouvoir déterminer l'origine de défauts ou de pannes.

En particulier, dans le cas de circuits intégrés défectueux, il convient de pouvoir déterminer celui ou ceux des composants constituant le circuit intégré qui présentent une anomalie de fonctionnement.

Le nombre des composants entrant dans la constitution d'un circuit intégré est généralement très élevé de sorte que la localisation du ou des quelques éléments défectueux est très délicate à effectuer parmi la multitude d'éléments d'où la panne peut résulter.

On connaît actuellement un procédé d'analyse de circuits intégrés connu sous le terme anglais de "back tracking". Pour la mise en œuvre de ce procédé, on utilise une installation de tests, permettant à l'aide de sondes matérielles ou virtuelles d'effectuer des relevés des signaux circulant en différents points de mesure du circuit.

A partir de la connaissance de la structure théorique du circuit intégré analysé, on détermine, en fonction d'une séquence de tests appliqués sur

les entrées du circuit, les signaux devant être mesurés aux différents points de mesure du circuit.

Pour localiser les éléments défectueux dans le circuit, on considère d'abord une sortie défectueuse du circuit puis on remonte depuis cette sortie vers les entrées en testant de proche en proche chacun des points de mesure successifs. Tant que les mesures effectuées aux différents points révèlent des signaux incorrects par rapport aux signaux théoriques devant être obtenus, on déduit que les éléments défectueux du circuit sont en amont du point de mesure. Dès qu'on obtient, en un point de mesure, des signaux corrects, on déduit que l'élément défectueux est situé entre le point de mesure où des signaux corrects sont obtenus et le point de mesure précédent où des signaux incorrects de mesure ont été obtenus.

Chaque mesure effectuée réellement sur le circuit intégré nécessite un temps considérable pouvant aller de quelques secondes si le point de mesure est en surface à 5 à 10 minutes si le point de mesure est situé sur une couche profonde du circuit intégré et qu'un accès matériel préalable doit être ménagé à l'aide, par exemple, d'un faisceau d'ions focalisé.

On conçoit que, avec la méthode utilisée actuellement, il convient parfois de parcourir à rebours l'essentiel du circuit intégré si l'élément défectueux est très proche d'une entrée du circuit. Du fait de la complexité des circuits et des nombreuses ramifications le constituant, la recherche de l'élément défectueux peut s'avérer extrêmement longue.

L'invention a pour but de proposer un procédé de détection d'erreurs dans un circuit permettant une localisation plus rapide des zones défectueuses, tout en conservant une grande fiabilité dans cette localisation.

A cet effet, l'invention a pour objet un procédé du type précité, caractérisé en ce qu'il comporte initialement :

- une étape de modélisation de l'agencement théorique du circuit intégré, sous forme d'au moins un graphe comportant un ensemble de nœuds et d'arcs orientés des entrées du circuit vers les sorties du circuit ;

- considérer comme un sous-graphe de recherche , un sous-graphe dont le nœud formant sommet correspond à un point de mesure défaillant ;

et en ce que, pour la recherche de l'élément défectueux, il comporte les étapes de :

- affecter à chaque nœud du sous-graphe de recherche considéré, une variable caractéristique dépendant de la structure du sous-graphe de recherche ;

- considérer comme point de mesure, le point de mesure correspondant à un nœud du sous-graphe considéré, obtenu par application d'un critère prédéterminé portant sur les variables caractéristiques de l'ensemble des nœuds du sous-graphe de recherche considéré ;

- effectuer un test du point de mesure considéré ;

- considérer comme nouveau sous-graphe de recherche :

soit, le sous-graphe de recherche préalablement considéré en excluant le nœud correspondant au point de mesure testé et tous ses nœuds parents, si le point de mesure est satisfaisant

soit, un sous-graphe dont le nœud correspondant au point de mesure est le sommet , si le point de mesure est défaillant ; et

- rechercher, dans le nouveau sous-graphe de recherche considéré l'élément défectueux, jusqu'à vérification d'un critère d'arrêt prédéterminé.

Suivant des modes particuliers de mise en œuvre, le procédé comporte l'une ou plusieurs des caractéristiques suivantes :

- lors de l'étape initiale de modélisation de l'agencement théorique du circuit intégré, le circuit est modélisé sous forme d'un arbre par création éventuelle de nœuds virtuels lorsque un même nœud est le parent de au moins deux nœuds, eux-mêmes parents d'un même nœud ;

- il comporte, après vérification du critère d'arrêt prédéterminé, les étapes de :

. évaluer dans le ou chaque dernier sous-graphe de recherche, si pour chaque nœud virtuel correspondant à un point de mesure défaillant, le nœud jumeau associé audit nœud virtuel est un nœud du même sous graphe correspondant également à un point de mesure défaillant ; et

. considérer alors le ou chaque sous-groupe pour lequel la condition est vérifiée comme correspondant à une partie du circuit intégré comportant au moins un élément défectueux ;

- ladite variable caractéristique propre à chaque nœud est le nombre d'ancêtres de ce nœud dans le sous-graphe de recherche considéré ;

- ledit critère prédéterminé est adapté pour déterminer le nœud dont le nombre d'ancêtres est sensiblement égal au nombre moyen d'ancêtres par nœud dans le sous-arbre de recherche considéré ;

- il comporte une étape d'affectation à chaque nœud, d'un indicateur de conformité initialement fixé à un état défaillant ; et

pour la détermination du nouveau sous-graphe de recherche à considérer, il comporte les étapes de :

. fixer l'indicateur de conformité du nœud correspondant au point de mesure testé et de tous ses nœuds parents à un état satisfaisant, si le point de mesure testé est satisfaisant ; et

. considérer comme nouveau sous-graphe de recherche, le sous-graphe inclus dans le sous-graphe de recherche précédent et comportant les seuls nœuds dont l'indicateur de conformité est fixé à l'état défaillant ; et

- le sous-graphe de recherche initialement considéré est formé de l'intersection des sous-graphes ayant chacun pour sommet un nœud correspondant à une sortie défaillante du circuit intégré.

L'invention a également pour objet un dispositif de localisation d'un élément défectueux dans un circuit intégré dont l'agencement théorique est connu, du type comportant des moyens pour effectuer une succession d'étapes consistant en :

- la détermination d'un point de mesure du circuit intégré ; et

- le test du point de mesure déterminé par mise en œuvre de :

- l'application d'une séquence de tests aux entrées du circuit intégré ;

- la mesure de signaux au point de mesure déterminé du circuit intégré, lors de l'application de la séquence de tests ; et

- l'évaluation du point de mesure par la comparaison des signaux mesurés avec des signaux théoriques devant être obtenus au

point de mesure déterminé pour évaluer si le point de mesure est défaillant ou satisfaisant ; et

des moyens pour déterminer la position de l'élément défectueux du circuit intégré à partir des évaluations effectuées aux différents points de mesure déterminés,

caractérisé en ce qu'il comporte :

- des moyens de modélisation initiale de l'agencement théorique du circuit intégré, sous forme d'au moins un graphe comportant un ensemble de nœuds et d'arcs orientés des entrées du circuit vers les sorties du circuit ;
- des moyens pour considérer initialement comme un sous-graphe de recherche , un sous-graphe dont le nœud formant sommet correspond à un point de mesure défaillant ;

et en ce que, pour la recherche de l'élément défectueux, il comporte des moyens pour :

- affecter à chaque nœud du sous-graphe de recherche considéré, une variable caractéristique dépendant de la structure du sous-graphe de recherche ;

- considérer comme point de mesure, le point de mesure correspondant à un nœud du sous-graphe considéré, obtenu par application d'un critère prédéterminé portant sur les variables caractéristiques de l'ensemble des nœuds du sous-graphe de recherche considéré ;

- effectuer un test du point de mesure considéré ;

- considérer comme nouveau sous-graphe de recherche :

soit, le sous-graphe de recherche préalablement considéré en excluant le nœud correspondant au point de mesure testé et tous ses nœuds parents, si le point de mesure est satisfaisant

soit, un sous-graphe dont le nœud correspondant au point de mesure est le sommet , si le point de mesure est défaillant ; et

- rechercher, dans le nouveau sous-graphe de recherche considéré l'élément défectueux, jusqu'à vérification d'un critère d'arrêt prédéterminé.

L'invention sera mieux comprise à la lecture de la description qui va suivre, donnée uniquement à titre d'exemple et faite en se référant aux dessins sur lesquels :

- la figure 1 est une vue schématique d'un synoptique général donnant l'interdépendance des données et des actions lors de la mise en œuvre d'un procédé selon invention ;

- la figure 2 est une vue plus détaillée de l'organigramme mis en œuvre dans l'installation dont le synoptique est donné sur la figure 1 ;

- les figures 3 et 4 sont des vues schématiques respectivement d'un circuit élémentaire et de sa représentation sous forme d'un graphe d'influence inter-nœuds ;

- la figure 5 une vue d'un exemple d'une zone défaillante d'une modélisation d'un circuit intégré ;

- les figures 6 et 7 sont des vues respectivement d'une cellule défaillante seule et de deux cellules défaillantes partageant la même sortie ;

- les figures 8A et 8B sont des vues schématiques respectivement d'un graphe comportant un cycle et d'un arbre résultant d'une modification du graphe par application du principe de coupure ;

- la figure 9 une vue schématique de cellules de l'arbre illustrant le principe du parcours d'un arbre pour son marquage ;

- la figure 10 est un organigramme explicitant le séquencement général du marquage par la fonction **marquage ()** ;

- la figure 11 est un organigramme explicitant la fonction **nœud_cherche_sorties( )** utilisée lors du marquage ;

- la figure 12 est un organigramme explicitant la fonction **sortie_cherche_nœuds( )** utilisée lors du marquage ;

- la figure 13 est une vue schématique de la construction d'un sous-ensemble minimal ;

- la figure 14 est une vue schématique d'intersections non connexes entre sous-graphes de l'arbre de référence ;

- la figure 15 est une vue schématique donnant un exemple d'une intersection de deux arbres ;

- la figure 16 est une vue schématique d'un arbre donnant un exemple de numérotation ;

- la figure 17 est un organigramme explicitant l'algorithme de recherche d'éléments défectueux par la fonction **dichotomie( )** ;

- la figure 18 est un organigramme explicitant le calcul et la localisation d'un nœud moyen par la fonction **chercher_noeud_moyen ( )** ;

- la figure 19 est un organigramme explicitant le calcul du nombre de nœuds et du cumul des ancêtres par la fonction **calcul_moyenne( )** ;

- les figures 20 et 21 sont des organigrammes explicitant les éléments modifiés de l'algorithme de la figure 19 pour la localisation d'un nœud moyen par la fonction **localiser_moyenne** () ;

- la figure 22 est une vue d'un arbre illustrant des exemples de résolutions autorisées et interdites ;

- la figure 23 est un organigramme explicitant la détermination des nœuds pouvant être déclarés sommet d'une zone défaillante par la fonction **analyser_cellules_restantes( )**.

- la figure 24 est un organigramme explicitant la construction de la liste des candidats par la fonction **construire_liste_candidats( )**

- la figure 25 est un organigramme explicitant l'analyse sous un nœud sommet par la fonction **analyser_sous_ce_sommet( )**.

- la figure 26 est un organigramme explicitant la construction de la liste des coupures par la fonction **lister_coupures_a_resoudre( )**.

- la figure 27 est un organigramme explicitant l'algorithme de résolution des coupures par la fonction **resoudre_coupures( )** ; et

- la figure 28 est un organigramme explicitant la recherche d'une résolution interne par la fonction **chercher_un_nœud( )**.

Sur la figure 1 est représenté le schéma synoptique des données et des actions mises en œuvre par une installation de contrôle d'un circuit intégré mettant en œuvre le procédé selon l'invention.

Le programme mis en œuvre par cette installation permet de déterminer progressivement, à partir du schéma théorique d'un circuit intégré, les points où des mesures physiques doivent être réalisées par mise en œuvre d'une séquence de tests appropriés.

Après chaque acquisition du résultat du test précédemment effectué, permettant de déterminer si le signal reçu au point de mesure est ou non correct, le procédé mis en œuvre détermine un nouveau point où un test doit être effectué à partir d'une nouvelle séquence de tests.

Au fur et à mesure de l'acquisition des résultats des tests proposés automatiquement, le procédé selon l'invention permet de localiser la région du circuit intégré dans laquelle se trouve l'élément défectueux puis finalement de localiser celui-ci de manière rapide.

Les séquences de tests mises en œuvre pour chacun des points de mesure du circuit intégré sont de tout type adapté et sont par exemple comparables à celles mises en œuvre dans un procédé de "back tracking". Les séquences de tests étant connues en soi, elles ne seront pas décrites dans la suite de la description.

De même, les moyens utilisés pour assurer la mesure d'un signal en un point de mesure déterminé du circuit intégré, lors de l'application d'une séquence de tests ne seront pas décrits en détail, ceux-ci étant de tout type adapté.

Le procédé selon l'invention peut être mis en œuvre avec une installation comportant, d'une part, des moyens de test tels qu'un microscope électronique à balayage permettant d'effectuer des relevés des signaux circulant en des points de mesure déterminés du circuit intégré lors de l'application d'une séquence de tests, et, d'autre part, une unité de traitement d'informations, telle qu'un micro-ordinateur mettant en œuvre un programme adapté déterminant, suivant le procédé de l'invention, les points du circuit où des mesures successives doivent être effectuées, et déduisant des résultats de mesures la position des éléments défectueux dans le circuit.

Sur la figure 1 sont illustrés les différents fichiers de données utilisés ainsi que les traitements opérés sur ceux-ci, afin de déterminer, d'une part, les points successifs où des mesures doivent être effectuées sur le circuit et, d'autre part, les éléments ou zones défectueux du circuit.

La structure générale du programme met en œuvre la notion de session.

Une session peut être définie comme un ensemble de fichiers cohérents relatifs à des conditions données pour les entrées et les paramètres du programme. Une session contient aussi l'ensemble des fichiers intermédiaires générés par le programme, nécessaires au fonctionnement, et à la mémorisation du numéro de la dernière étape de calcul effectuée.

Le contenu d'une session est défini dans un fichier texte dont le nom porte l'extension « .ses ». La racine du nom est définie librement par l'utilisateur.

Le chargement par le programme d'un fichier de session préexistant restaure le contexte de calcul propre à cette session (mêmes fichiers de données, mêmes paramètres de contrôle et résultats intermédiaires) et permet le redémarrage des calculs à l'endroit où ils avaient été interrompus.

Le séquencement des différentes opérations est effectué par une fonction **main( )**.

Le code de cette fonction se trouve dans un fichier « principal.c ».

Selon le principe du programme, les différentes opérations sont enchaînées de manière séquentielle.

On distingue cinq phases principales qui seront détaillées dans la suite :

- La phase d'interprétation de l'EDIF
- La phase de marquage
- La phase de numérotation (cette phase englobe la construction de l'arbre de référence, celle du sous-ensemble minimal et la création du sommet virtuel)
- La phase de localisation des fautes
- La phase de compte-rendu.

A l'issue de chacune de ces phases, les calculs peuvent être interrompus. Dans ce cas, tous les résultats intermédiaires nécessaires à la reprise ultérieure des calculs sont automatiquement archivés dans des fichiers de sauvegarde. Ces fichiers ont des noms comportant une extension « .data ». Le degré d'avancement des calculs est également archivé.

La reprise des calculs se fait de façon transparente, c'est-à-dire sans avoir à faire référence à ces fichiers intermédiaires, en chargeant simplement le fichier de session approprié.

Initialement, la structure théorique du circuit, c'est-à-dire la structure d'un circuit sans défaut, est stockée dans un fichier 100, qui est par exemple au format EDIF.

Ce fichier est traité d'abord lors d'une phase d'interprétation 102 permettant la mise en forme du fichier décrivant le circuit théorique sous forme d'un format interne propre à la mise en œuvre du procédé. La réalisation de l'interprétation 102 et le format propre à l'application du procédé seront décrits dans la suite de la description.

Généralement, l'interprétation consiste à modéliser le circuit théorique sous la forme d'un graphe, dont les nœuds correspondent aux entrées et aux sorties des différents composants du circuit intégré et dans lequel les composants sont représentés par des ensembles d'arcs orientés reliant les nœuds entre eux.

L'algorithme d'interprétation reçoit et adresse des données à une mémoire interne de session 104 assurant notamment un stockage temporaire du fichier contenant le graphe du circuit présenté suivant un format exploitable.

Cette mémoire 104 est reliée à un fichier de session 106 assurant un stockage des données de session sur un support permanent tel qu'un disque dur.

A l'issue de l'interprétation 102, le fichier de session est soumis à une phase de marquage 108 associée à d'autres traitements préliminaires.

La phase suivante notée 110 est dite "phase de construction d'un sous-ensemble minimal et de numérotation". Elle a notamment pour but la définition de sous-ensembles parmi les différents arbres modélisant le circuit. En particulier, lors de la phase 110, sont prises en compte deux séries d'informations à savoir la liste des sorties du circuit fonctionnant correctement, notée 112, et la liste des sorties du circuit ne fonctionnant pas correctement notée 114. A cet effet, les sorties du circuit sont testées à partir de

séquences de tests, comme cela sera expliqué dans la suite de la description.

A partir du fichier de session obtenu en sortie de l'étape 110, une phase 116 de chargement des listes est mise en œuvre. Cette phase prend en compte la liste des nœuds interdits stockés dans un fichier 118, la liste des niveaux de métal 120 précisant les couches métalliques sur lesquelles sont présents les différents éléments du circuit, ainsi qu'un fichier 122 comportant la liste des nœuds déjà testés.

La phase ultérieure 123 consiste en la localisation des composants ou groupes de composants défectueux, causant des fautes dans le fonctionnement du circuit. Selon l'invention, cette recherche des fautes s'effectue suivant un procédé de dichotomie pratiqué sur certains arbres particuliers du circuit, en tenant compte d'une pondération encore appelée numérotation de chacun des nœuds effectuée suivant une méthode prédéterminée.

Lors de la phase de dichotomie notée 124, est également mise en œuvre une phase d'analyse supplémentaire 126 dite "phase d'analyse après épuisement" permettant de compléter la recherche des fautes effectuée par dichotomie. Cette phase supplémentaire permet de tenir compte des contraintes de modélisation du circuit qui ont conduit à certaines modifications conventionnelles du graphe le représentant.

Enfin, les résultats sont mis à disposition de l'utilisateur lors de la phase ultime 128.

Comme représentée sur le schéma synoptique, chacune des phases de traitement 102, 108, 110, 116, 124, 126, 128 reçoit et adresse des données depuis et vers la mémoire interne de session 104. Cette dernière est de plus reliée à un ensemble de fichiers résultats 130, ainsi qu'à un journal de stockage 132. Ces derniers assurent respectivement la mise à disposition de l'utilisateur des résultats de l'analyse et l'archivage des commentaires imprimés à l'écran.

Sur la figure 2 est décrite plus en détail chacune des phases successives 102, 108, 110, 124 et 126 de mise en œuvre du procédé selon l'invention.

Ainsi, la description détaillée de l'algorithme total va être effectuée en regard de la figure 2, où chacune des étapes élémentaires sera décrite successivement en référence à d'autres figures illustratives encore plus détaillées.

La première étape de la phase initiale 102 de mise en forme du fichier décrivant le circuit intégré est désignée par la référence 202 sur la figure 2.

De manière générale, un circuit peut toujours être décrit comme un ensemble de cellules comportant des entrées et des sorties connectées par des nœuds. Le choix du niveau hiérarchique de définition d'une cellule est arbitraire. Il s'agit par exemple d'un sous-ensemble fonctionnel constituant une macro-cellule, d'une porte ou d'un transistor. De même, la notion d'entrées et de sorties du circuit à analyser peut être immédiatement transposée aux entrées et sorties d'un bloc interne, l'analyse portant alors sur ce bloc.

Avec le type de description utilisé, les signaux ne sont observés que sur les nœuds.

La description des cellules se réduit à l'influence qu'exercent leurs nœuds d'entrée sur leurs nœuds de sortie. La description initiale du circuit formée d'un ensemble de cellules et de nœuds, peut donc être transformée en un graphe d'influence inter-nœuds en remplaçant chaque cellule, dont un exemple est donné sur la figure 3, par un ensemble d'arcs orientés tel qu'illustré sur la figure 4.

Sur la figure 3, est représentée une cellule 300 constituée par exemple d'une porte ET. Cette cellule comporte deux entrées E1 et E2 reliées à des nœuds d'entrée 302 et 304 respectivement. Ces nœuds d'entrée sont reliés aux sorties d'autres cellules non représentées du circuit intégré. La sortie, notée S, de la cellule 300 est reliée à un nœud 306 auxquelles sont reliées les entrées d'autres cellules non représentées du circuit intégré.

Sur la figure 4, la cellule 300 est remplacée par deux arcs orientés 402 et 404 reliant respectivement les nœuds d'entrée 302 et 304 aux nœuds de sortie 306 de la cellule.

Ainsi, la cellule 300 est modélisée simplement par les deux arcs orientés 402 et 404.

Grâce à une telle modélisation de toutes les cellules, la description totale du circuit prend ainsi la forme d'un graphe.

Dans une telle modélisation du circuit et comme illustré sur la figure 5, une zone défaillante 500 est un sous-graphe et même, après application de modifications qui seront explicitées ultérieurement, un sous-arbre.

Sur cette figure apparaissent plusieurs types de nœuds.

Les nœuds mauvais sont désignés par un rond au centre duquel figure une croix. Les nœuds mauvais sont les nœuds pour lesquels, pour une séquence de tests donnée appliquée à l'entrée du circuit, est obtenu un signal différent du signal théorique devant être obtenu pour ce point de mesure.

Certains nœuds sont des nœuds non testables. Il s'agit par exemple des nœuds situés sur des couches profondes du circuit intégré auxquels l'accès est impossible ou difficile. Ces nœuds sont désignés par un rond à l'intérieur duquel figure un point d'interrogation.

Les nœuds bons, par opposition aux nœuds mauvais, sont des nœuds où pour une séquence de tests donnée, le signal mesuré correspond au signal théorique devant être obtenu. Ces nœuds bons sont désignés par un rond à l'intérieur duquel figure un autre rond.

Dans l'exemple de la figure 5, la zone défaillante 500 est constituée d'un sous-arbre dont le sommet 502 est un nœud mauvais. Ce nœud mauvais 502 est commandé par deux nœuds non testables 503 et 504. Le premier nœud non testable 503 est relié à deux nœuds bons 506 et 508. Le second nœud non testable 504 est relié à un nœud bon 510 et à un nœud non testable 512 lui-même relié à un nœud bon 514.

De manière générale, un sous-arbre constituant une zone défaillante a les propriétés suivantes :

- Le sommet est un nœud sur lequel le signal est mauvais.
- Toutes les terminaisons sont des nœuds sur lesquels le signal est bon.
- Il peut exister des nœuds non testables entre le sommet et les terminaisons. Dans le cas contraire, il s'agit d'une cellule défaillante ou de cellules défaillantes dont les sorties sont connectées au même nœud.

Un exemple de cellule défaillante seule est représenté sur la figure 6. Un exemple de deux cellules défaillantes partageant la même sortie est décrit sur la figure 7. Sur ces figures, les conventions de la figure 5 sont utilisées.

Dans la pratique, l'étape d'interprétation 202 consiste à transformer le fichier initial de description du circuit, au format Edif, par exemple, en un fichier d'un format déterminé propre à la mise en œuvre du procédé (fichier dont le nom porte le suffixe « .parsed »).

Suivant la nature du fichier initial (repérée par l'un des suffixes « .edn », « .edf » ou « .edo »), le programme lance l'interpréteur approprié.

Ces programmes d'interprétation sont utilisés comme des commandes du système d'exploitation par la fonction « **system( )** » du langage C.

Dans les cas où la description initiale n'est pas conforme aux formats des interpréteurs disponibles, il est toujours possible d'utiliser directement une description au format interne (format « .parsed »).

Le format de ces fichiers internes est donné sur la table 1.

Les contraintes conventionnelles à respecter sont les suivantes :

- Il s'agit d'un fichier texte.
- On ne doit pas sauter de ligne.
- Les différents blocs doivent être placés dans l'ordre suivant : les blocs décrivant les sorties ; les blocs décrivant les entrées ; les blocs décrivant les cellules internes et les blocs décrivant les nœuds.
- La numérotation des sorties commence à zéro et se termine à N.
- La numérotation des entrées commence à N+1 et se termine à (Nombre_de_broches – 1).
- La numérotation des cellules internes (instances) commence à (Nombre_de_broches) et se termine à (Nombre_de_broches + Nombre_de_cellules_internes – 1).
- Les nœuds sont numérotés de zéro à (Nombre_de_nœuds – 1).
- La numérotation des nœuds dans les blocs relatifs aux sorties, aux entrées et aux cellules instanciées doit être cohérente avec la numérotation des blocs décrivant les nœuds.

- Les numéros des cellules dans les blocs décrivant les nœuds doivent être cohérents avec la numérotation des cellules instanciées, la numérotation des entrées, ainsi que la numérotation des sorties effectuée en amont dans le fichier.

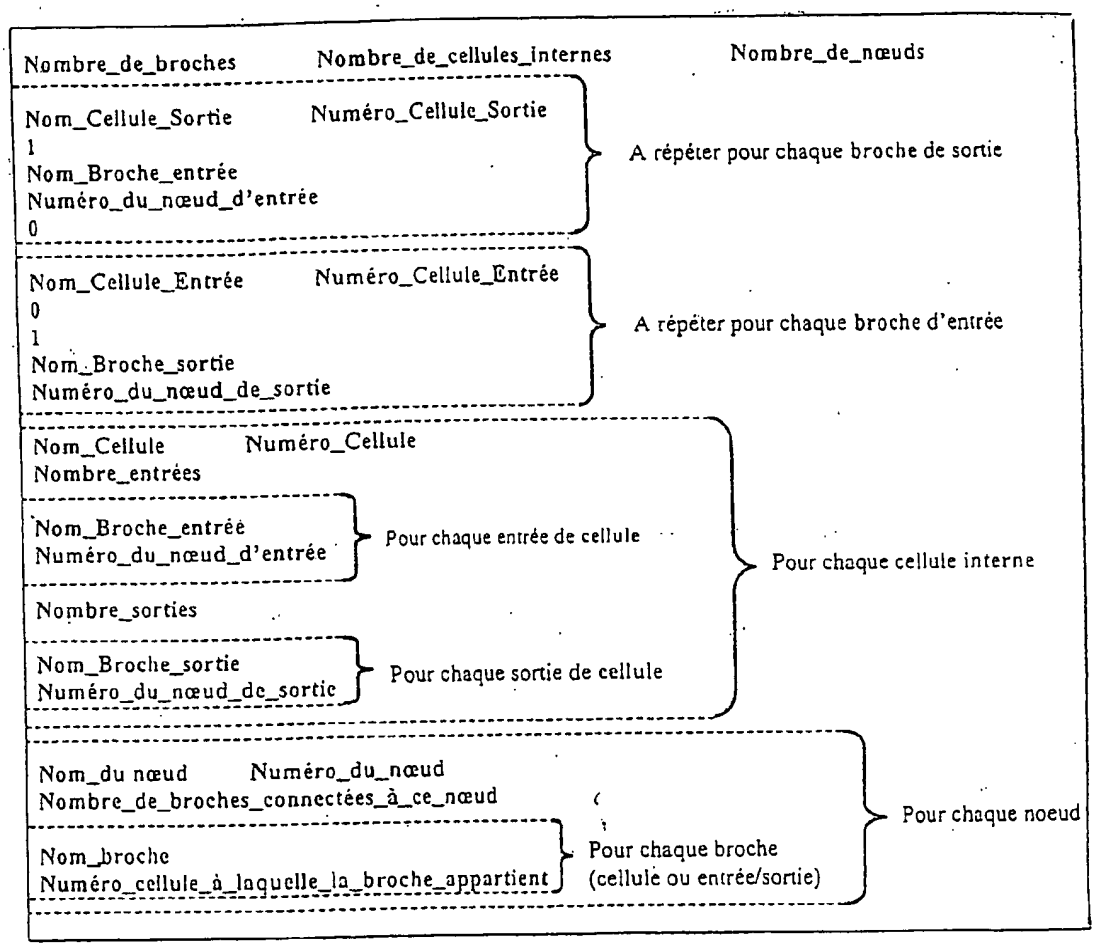


Table 1 : Structure d'un fichier au format de description interne (« .parsed »)

Trois listes fondamentales sont construites à partir de ce fichier. Il s'agit de :

- Une liste chaînée de structures de type NŒUD pour décrire les nœuds ;
- Une liste chaînée de structures de type CELLULE pour décrire le brochage ;

- Une liste chaînée de structures de type CELLULE pour décrire les cellules instanciées ;

La définition de ces structures se trouve dans un fichier d'en-tête « structures.h ».

Toutes les autres structures de données construites par la suite contiennent, en général, des pointeurs vers les éléments de ces listes de manière à éviter au maximum la duplication des informations.

La première étape 204 de la phase de marquage et de traitements préliminaires 108 consiste à trouver, pour chaque sortie du circuit intégré, l'ensemble des nœuds susceptibles de l'influencer. Ces ensembles de nœuds sont appelés « cônes d'influence ». Simultanément est effectué un marquage des nœuds comme cela sera décrit dans la suite de la description.

A l'issue de cette étape 204, il est possible d'énumérer, pour chaque nœud du circuit, l'ensemble des sorties du circuit que le nœud est susceptible d'influencer. C'est à partir des cônes d'influence, affectés à chaque nœud, que s'effectuent les différentes opérations d'intersection et d'exclusion présentées dans la suite de la description.

Au début de l'étape 204 de construction des cônes d'influence, on effectue un certain nombre de calculs préliminaires relatifs aux sorties. Cela consiste à :

- Compter le nombre de sorties ;
- Compter le nombre de groupes de marqueurs ;
- Remplir les tableaux communs associés aux sorties.

Ces tableaux communs sont :

- Le tableau donnant le numéro de groupe de chaque sortie : **groupeSortie[ ]** ;
- Le tableau donnant le rang de chaque sortie : **rangSortie[ ]** ;
- Le tableau donnant la valeur numérique du marqueur pour chaque sortie : **marqueurSortie[ ]** ;
- Le tableau contenant les adresses des structures relatives à chaque sortie : **adresseSortie[ ]** ;
- Le tableau des noms des sorties **nomSortie[ ]** ;

- Le tableau contenant le nombre total de nœuds dans le cône relatif à chaque sortie : **totalSortie[ ]**.

L'utilité de ces différents tableaux apparaît après la lecture des paragraphes suivants relatifs à la méthode de marquage des nœuds.

Pour construire le cône d'influence d'une sortie, on part de cette sortie et on parcourt le graphe d'influence avec un algorithme standard de parcours des arbres tel qu'un algorithme à « ordre préfixé ».

Comme le graphe d'influence n'est pas un arbre, suivant la nature du circuit initial, ce graphe peut comporter des cycles dus à la présence dans le circuit de boucles séquentielles ou combinatoires.

Lorsqu'on s'engage dans un tel cycle, on revient à un moment donné sur un nœud déjà analysé. L'algorithme est adapté pour détecter le retour sur un nœud. Quand cet événement se produit, l'algorithme crée une terminaison fictive, dénommée "coupure", et rebrousse chemin.

Suivant le principe des coupures, on effectue des coupures quand on revient sur des nœuds déjà analysés. La notion de coupure est illustrée sur les figures 8A et 8B.

Sur la figure 8A est représenté un graphe d'influence ne présentant pas la forme d'un arbre puisque l'un des nœuds noté 802 est le parent de deux nœuds 804 et 806, ces derniers étant eux-mêmes les parents d'un même nœud 808. Ainsi, les nœuds 802, 804, 808 et 806 forment une boucle combinatoire.

Afin de créer un arbre à partir du graphe de la figure 8A, et comme illustré sur la figure 8B, un nœud virtuel 810, constituant une coupure est ajouté comme parent du nœud 806. Dans la suite des dessins, les coupures sont représentées par un rond renfermant un carré.

Le nœud 802 est conservé comme parent du nœud 804. Ainsi, le nœud 810 constitue un nœud jumeau virtuel du nœud 802. L'arc orienté issu du nœud 802 et pointant sur le nœud 806 est supprimé dans la structure de données. Il est indiqué pour mémoire en traits pointillés sur la figure 8B.

Dans la structure de données, un champ « parent » de la structure décrivant le nœud 810 pointe sur le nœud jumeau 802 dans la liste des

nœuds. En opérant de cette manière, on obtient un arbre puisque tous les cycles sont ouverts.

Les cônes d'influence sont donc des sous-graphes du graphe d'influence munis de coupures appropriées.

Ils ont une structure d'arbre dont le sommet est une sortie et les terminaisons sont soit des entrées soit des coupures.

Simultanément à la définition des cônes d'influence est effectué un marquage des nœuds.

Le but du marquage est d'enregistrer pour chaque nœud les sorties du composant qu'il influence.

Cet enregistrement se fait en agissant sur un "marqueur" associé à chaque nœud.

Chaque fois qu'on trouve le long du parcours un nœud nouveau, on attribue à la variable "marqueur", qui est affectée au nœud, le numéro de la sortie dont on construit le cône d'influence.

Ces marqueurs sont des tableaux de nombres entiers dont le type (TYPEMASQUE) est un type prédéfini dans l'en-tête « structure.h ». Le champ « marqueur » dans chaque structure « NŒUD » est un pointeur sur le premier élément de ce tableau.

Chacun des bits de ces entiers est utilisé pour mémoriser si une sortie est influençable ou non par ce nœud. Si le bit correspondant à la sortie d'indice n est mis à 1, le nœud influence la sortie n. Sinon ce bit est laissé à zéro.

Comme les entiers sont limités en nombre de bits, il faut partager l'ensemble des sorties en groupes et affecter un entier à chaque groupe pour la mémorisation des sorties du groupe. Le nombre de sorties codables est ainsi illimité.

Dans un premier temps, le programme compte les broches de sortie.

Soit « nombreDeSorties » le nombre des broches de sorties.

Le nombre de bits par entier est donné par une instruction du langage C :

NOMBREDEBITS = sizeof(TYPEMASQUE) * 8.

Le nombre de groupes est donc donné par :

nombreDeGroupes = PartieEntière((nombreDeSorties – 1) / NOMBREDEBITS) + 1

Ce nombre de groupes sert à dimensionner les tableaux de marquage.

Le groupe de la sortie n est déterminé par : groupeSortie[n] = PartieEntière(n / NOMBREDEBITS).

On définit également le rang de la sortie n par : rangSortie[n] = n Modulo(NOMBREDEBITS).

Une sortie donnée est donc repérée par un couple (groupe, rang).

Pour mettre à 1 le bit correspondant à la sortie numéro n, on ajoute « 2 à la puissance rangSortie[n]» à l'entier du tableau correspondant au groupe de la sortie n.

Pour chaque sortie, la valeur correspondante de cet incrément du marqueur est calculée dans les calculs préliminaires. Ces valeurs sont mémorisées dans le tableau commun « marqueurSortie[] ».

Si « node » est le pointeur contenant l'adresse du nœud à traiter, l'activation de son marqueur pour la sortie « n » se fait donc avec l'instruction C suivante :

node->marqueur[groupeSortie[n]] += marqueurSortie[n] ;

Pour que ce mécanisme fonctionne, il faut que chaque nœud pouvant être marqué pour la sortie n ne le soit qu'une seule fois. Avant de marquer un nœud pour la sortie n, il faut donc tester si ce nœud a déjà été marqué ou non pour cette sortie. Pour ce faire, on opère de la façon suivante :

L'adresse « node » du nœud et le numéro n de la sortie étant donnés, on détermine le groupe correspondant à cette sortie en lisant le tableau groupeSortie[].

On utilise ce numéro de groupe pour localiser l'élément correspondant à cette sortie dans le tableau marqueur associé à ce nœud.

On applique le ET logique entre cet élément du tableau et la valeur contenue dans le tableau commun marqueurSortie[] pour ce numéro de sortie, c'est-à-dire :

test = (node->marqueur[groupeSortie[n]]) & (marqueurSortie[n]) ;

La valeur de test est différente de zéro si le bit correspondant à la sortie n est déjà à la valeur 1 dans le marqueur associé au nœud.

L'attribution des valeurs correctes aux marqueurs associés à chaque nœud est effectuée en exécutant l'algorithme de marquage dont le principe est exposé ci-dessous

Cet algorithme garantit que tout nœud est marqué une fois et une seule pour toutes les sorties qu'il est capable d'influencer.

Cet algorithme est adapté au cas des marqueurs vectoriels, exposé ci-dessus. Il est également adapté pour la construction de l'arbre de référence, comme cela sera exposé dans la suite.

Le principe de l'algorithme est de parcourir le cône d'influence du circuit depuis chaque sortie avec un algorithme standard de parcours des arbres tel qu'un algorithme à ordre « préfixé ». Chaque fois qu'un nœud peut être marqué, on vérifie s'il ne l'a pas déjà été. S'il a déjà été marqué, l'algorithme considère qu'il a atteint une terminaison et il rebrousse chemin.

Pendant le marquage, on compte également le nombre total de nœuds contenus dans chaque cône d'influence. Ces totaux sont stockés dans le tableau commun « totalSortie[ ] » .

La valeur « zéro » du paramètre « mode » indique à l'algorithme qu'il doit effectuer le marquage des nœuds. La valeur « un » du paramètre « mode » indique à l'algorithme qu'il doit effectuer la construction de l'arbre de référence. La construction de l'arbre de référence sera explicitée dans la suite de la description.

Le parcours du graphe s'effectue par l'appel récursif de deux fonctions illustrées schématiquement sur la figure 9.

L'enchaînement des différentes étapes de l'algorithme de marquage objet de la fonction **marquage ( )** est exposée sur la figure 10. Ces différentes étapes font appel aux fonctions ci-dessous exposées qui seront décrites en regard des figures 11 et 12.

- **La fonction nœud_cherche_sorties( )** (figure 11) : cette fonction est exécutée lors de l'analyse de l'environnement d'un nœud. Elle recherche les différentes sorties de cellules connectées à un nœud donné et pour chacune de ces sorties appelle la fonction **sortie_cherche_nœuds( )**.

- **La fonction sortie_cherche_nœuds( )** (figure 12) : cette fonction est exécutée lors de l'analyse des entrées de la cellule dont la sortie à été sélectionnée par la fonction **nœud_cherche_sorties( )**. Pour chacune des broches d'entrée de cette cellule, elle marque, s'il y a lieu, le nœud où elle est connectée et fait un appel à la fonction **nœud_cherche_sorties( )** depuis ce nouveau nœud.

Sur la figure 9, sont représentées deux cellules 902 et 904 dont la sortie est reliée à un même nœud 908 relié à une entrée d'une autre cellule 910. De même, deux entrées des cellules 902 et 904 sont reliées à un même nœud 906.

En initialisant le procédé sur le nœud 908, l'application de la fonction **nœud_cherche_sorties ()** permet d'identifier une des sorties des cellules 902 et 904. L'application ultérieure de la fonction **sortie_cherche_nœuds ()** permet de trouver le nœud 906 auquel sont connectées des entrées des cellules 902 et 904.

On conçoit que l'application récursive des deux fonctions permet de parcourir l'ensemble du graphe.

Sur la figure 10, est précisé l'organigramme de la fonction marquage. Sur cette figure, la variable n est utilisée pour parcourir les sorties du circuit intégré à tester. Initialement, à l'étape 1000, la variable n est fixée à 0.

Tant qu'il reste des sorties du circuit intégré à parcourir, la procédure de marquage est mise en œuvre. A cet effet, un test est effectué à l'étape 1002 pour comparer la variable n au nombre total de sorties du circuit intégré.

A l'étape 1004, est déterminé le nœud de la modélisation du circuit intégré associé à la sortie n considérée.

A l'étape 1006, ce nœud est marqué pour la sortie n.

A l'étape 1008, la fonction **nœud_cherche_sorties ()** est appliquée pour le nœud considéré. Ainsi, par l'application successive des fonctions récusives **nœud_cherche_sorties ()** et **sortie_cherche_nœuds ()**, l'ensemble de l'arbre associé à la sortie est parcouru par application des algorithmes décrits en regard des figures 11 et 12.

Après épuisement de l'arbre associé à la sortie n, la variable n est incrémentée à l'étape 1010. Le test opéré à l'étape 1002 est alors remis en œuvre. La fonction marquage s'achève après que toutes les sorties du circuit intégré ont été traitées.

La fonction **nœud_cherche_sorties ()** dont l'organigramme est présenté sur la figure 11, reçoit en entrée le nœud 1102 depuis lequel elle opère, le nom de la cellule précédente 1104, le numéro de la sortie du circuit intégré 1106 dont l'arbre est en cours de marquage ainsi que le mode de fonctionnement de la fonction 1108. Pour le marquage, et comme indiqué précédemment, le mode est fixé à 0.

A l'étape initiale 1110, une broche connectée au nœud fourni en 1102 est choisie. Il est vérifié à l'étape 1112 que cette broche existe. Si tel est le cas, à l'étape 1114, il est vérifié que la broche est connectée à la cellule précédente. Si tel est le cas, l'étape 1110 est remise en œuvre. Si la réponse au test effectué à l'étape 1114 est négative, il est vérifié à l'étape 1116 que la broche considérée est une broche de sortie. Si tel n'est pas le cas, une autre broche est choisie à l'étape 1110.

Si la broche considérée est une broche de sortie, l'étape 1118 est mise en œuvre. Celle-ci consiste en l'appel de la fonction **sortie_cherche_nœuds ()** associés aux paramètres correspondants.

A l'issue de la fonction **sortie_cherche_nœuds ()**, une nouvelle broche est choisie à l'étape 1110.

Lorsque appelée depuis l'étape 1118 et comme illustré sur la figure 12, la fonction **sortie_cherche_noeuds ()** dispose comme variable d'entrée du pointeur de la cellule considérée 1202, du numéro de sortie du circuit intégré considérée 1204, du nœud de départ 1206, ainsi que du mode 1208. Ce dernier est fixé à 0 pour le marquage.

A l'étape 1210, il est d'abord vérifié que la cellule est une cellule d'entrée ou de sortie. Si tel est le cas, l'algorithme rebrousse chemin. Si tel n'est pas le cas, l'étape 1212 est mise en œuvre au cours de laquelle une broche d'entrée de la nouvelle cellule est choisie.

Il est vérifié à l'étape 1214 que cette broche d'entrée existe. Si elle existe réellement, il est vérifié à l'étape 1216 que le nœud qui est connecté

est déjà marqué. Si celui-ci est déjà marqué, et que le mode est à 0 lors du test effectué à l'étape 1218, une nouvelle broche d'entrée est choisie par une nouvelle mise en œuvre de l'étape 1212.

Si le nœud considéré n'est pas déjà marqué, et que le mode est à 0 lors d'une vérification de l'étape 1220, ce nœud est marqué pour la sortie n considérée à l'étape 1222 et le compteur de nœuds de la sortie n est incrémenté. Enfin, à l'étape 1224, le choix d'une nouvelle broche d'entrée est effectué par appel de la fonction **nœud_cherche_sorties ()**.

Les appels récursifs réciproques des fonctions **sorties_cherche_nœuds ()** et **nœud_cherche_sorties ()** permettent le parcours total des arbres associés à chacune des sorties, l'ensemble des sorties étant balayé par l'application de l'algorithme de marquage de la figure 10.

Ainsi s'achève la phase de marquage 108. A l'issue de celle-ci, commence la phase 110 de construction du sous-ensemble minimal et de numérotation.

Pour la construction du sous-ensemble minimal, l'état de fonctionnement des sorties du circuit à analyser doit être connu.

A cet effet, une séquence de tests appropriés est préalablement appliquée sur les entrées du circuit à analyser, afin de déterminer parmi les sorties du circuit celles sur lesquelles apparaît un signal satisfaisant ou correct et celles sur lesquelles apparaît un signal défaillant ou incorrect.

Ainsi, les sorties du circuit sont réparties en deux groupes. Le premier groupe des sorties satisfaisantes, est défini dans une liste dénommée SOK (Acronyme pour Sortie OK).

Les sorties du circuit considérées comme défaillantes sont regroupées dans une seconde liste dénommée SNOK (Acronyme pour Sortie Non OK).

Ces listes sont désignées par les références 112 et 114 dans l'organigramme de la Figure 1.

En résumé, à ce stade, les données d'entrée du programme sont :

- Le fichier de description du circuit sous la forme d'un ensemble de cellules et de nœuds d'interconnexion. Si ce fichier est à l'un des

formats Edif (extensions « .edf , .edn ou .edo »), le programme lance automatiquement un interpréteur (parser). Le circuit peut également être décrit en utilisant le format interne de description (extension « .parsed »). Ce dernier format permet d'accéder à un niveau hiérarchique arbitraire de définition des cellules sans passer par les contraintes de l'Edif.

- La liste des sorties trouvées défaillantes à un moment donné de la séquence de test (SNOK).
- Une liste (facultative) de sorties qui n'ont jamais été trouvées défaillantes pendant le déroulement de la séquence de test (SOK).

A l'étape 212, l'algorithme détermine l'arbre de référence, c'est-à-dire l'arbre dans lequel les fautes du circuit seront recherchées.

Parmi les cônes relatifs aux sorties SNOK, il en existe au moins un qui contient le plus petit nombre de nœuds. Ce cône particulier est le cône de référence. C'est dans ce cône que seront recherchées les fautes. Ceci est exposé dans les paragraphes qui suivent.

Le programme sélectionne, pour la sortie de référence, la sortie défaillante dont le cône d'influence contient le plus petit nombre de nœuds. Ces nombres de nœuds sont contenus dans le tableau commun totalSorties[].

Un fois connue la sortie de référence, la fonction **nœud_cherche_sorties( )** est exécutée avec la valeur 1 pour le paramètre « mode ».

Le mécanisme de construction de l'arbre de référence est analogue à celui du marquage.

Cependant :

- Au lieu d'utiliser des tableaux de marqueurs, on utilise un champ scalaire « référence » de chaque structure NŒUD pour mémoriser l'appartenance d'un nœud à l'arbre de référence.
- On affecte à chaque nœud de l'arbre de référence une liste de parents. Le champ « liste_parents » dans chaque structure NŒUD est un pointeur sur le début de la liste de parents associée à ce nœud. Un élément de liste de parents contient un pointeur vers le nœud parent. Un pointeur sur

la cellule liant le nœud à ce parent et un témoin indiquant si ce parent est une coupure ou non.

Les éléments de liste de parents sont décrits dans le fichier d'en-tête « structures.h ».

Les étapes spécifiques à la construction de l'arbre de référence figurent en gras sur la figure 12.

En particulier, lors de la construction de l'arbre de référence, le test effectué à l'étape 1216 détermine si le nœud connecté à la broche choisie a été ou non référencée. Si tel est le cas, après l'étape 1218 où il est vérifié que le mode est égal à 1, l'étape 1240 est mise en œuvre. Elle consiste à ajouter un parent au nœud de départ en tant que coupure. A l'issue de cette étape, une nouvelle broche d'entrée est choisie à l'étape 1212.

Si lors du test effectué à l'étape 1216, il est constaté que le nœud qui est connecté à la broche d'entrée n'a pas déjà été référencé, et que le mode est égal à 1 lors du test effectué à l'étape 1220, l'étape 1242 est mise en œuvre. Lors de cette étape, le nœud considéré est référencé et ce nœud est ajouté à la liste des parents du nœud de départ.

A l'issue de cette étape, l'appel à la fonction **nœud_cherche_sorties** () est à nouveau effectué à l'étape 1224.

Le sommet de l'arbre de référence est le nœud d'entrée de la cellule de sortie associée à la sortie de référence.

La décomposition du graphe d'influence en arbres n'est pas unique. Elle dépend, en particulier, de l'ordre d'enregistrement des parents dans les listes de parents. Cependant ceci ne semble pas induire de problème particulier au niveau de la méthode.

Les étapes suivantes, notées 214 à 220, de la phase 110 de construction du sous-ensemble minimal consistent à construire le plus petit sous-ensemble de nœuds contenant les pannes et dans lequel s'effectuera la recherche proprement dite de fautes. Ce sous-ensemble est dénommé "sous-ensemble minimal".

Le sous-ensemble minimal est un sous-graphe de l'arbre de référence. Sa méthode de construction est donnée ci-dessous.

Lorsque une ou plusieurs fautes affectent simultanément un groupe de sorties, leur position est à rechercher dans le sous-ensemble formé par l'intersection des cônes d'influence de ces sorties.

Ce sous-ensemble est noté : $\bigcap_i SNOKi$

où SNOKi est le cône d'influence de la sortie d'indice i de la liste SNOK.

Un sous-ensemble constitué par la réunion des cônes d'influence de sorties non défaillantes correspond à un ensemble de nœuds qui ne sont pas des points de propagation de fautes.

Un sous-ensemble de ce type est noté : $\bigcup_j SOKj$

où SOKj est le cône d'influence de la sortie d'indice j de la liste SOK.

Un sous-ensemble $\bigcup_j SOKj$ a, en général, une intersection non vide avec le sous-ensemble $\bigcap_i SNOKi$

On peut donc réduire la recherche en éliminant de $\bigcap_i SNOKi$ cette intersection.

On obtient alors le "sous-ensemble minimal" noté SM dans lequel seront effectuées toutes les recherches de fautes.

Du point de vue formel, on a :

$$SM = \bigcap_i SNOKi - (\bigcap_i SNOKi \cap \bigcup_j SOKj)$$

Le sous-ensemble SM est illustré sur la figure 13 en considérant deux cônes notés 1302 et 1304 correspondant à des sorties correctes SOK1 et SOK2 et deux cônes notés 1306 et 1308 correspondant à des sorties incorrectes SNOK1 et SNOK2.

Le sous-ensemble minimal SM trouvé a la structure d'un ensemble d'arbres disjoints. Les sommets de ces arbres sont réunis à un nœud virtuel de manière à ce que les algorithmes de recherche appliqués par la suite n'aient à manipuler qu'un seul arbre.

Pour une liste donnée de sorties défaillantes, la construction de l'intersection $\bigcap_i SNOKi$ se fait en recherchant l'ensemble des nœuds qui influencent simultanément toutes les sorties de la liste.

Il peut donc se faire que l'intersection des sorties défaillantes conduise à un sous-ensemble vide.

Tel est le cas si les sorties défaillantes ont des cônes disjoints ou bien si l'intersection est non connexe comme illustré sur la figure 14. Dans ce cas, il y a plusieurs zones défaillantes indépendantes.

Dans le cas particulier de la figure 14, les cônes 1402, 1406, 1408, et 1410 sont chacun associés à une sortie défaillante du circuit intégré. Le cône 1402 est disjoint des cônes 1406 à 1410, ces trois derniers cônes présentant des intersections.

On constate, par les zones hachurées sur la figure 14, que l'intersection des cônes 1406 à 1410 forme deux ensembles disjoints notés 1412 et 1414 alors que le cône 1402 constitue lui-même une zone défaillante dont l'étendue ne peut être réduite puisque celui-ci ne possède pas d'intersection avec les autres cônes associés à des sorties défaillantes.

Le programme est adapté pour détecter ces cas particuliers et fractionne la liste des sorties défaillantes de manière à ce que pour chaque fraction de liste l'intersection trouvée par le programme soit connexe et minimale.

Pour explorer correctement ces zones, l'opérateur doit re-exécuter le programme sur chacune des fractions de liste proposées.

Dans un sous-ensemble connexe, le programme est adapté pour traiter des fautes multiples, sauf si ces fautes interagissent et se corrigent mutuellement, par exemple.

Afin de construire le sous-ensemble minimal, on construit d'abord à l'étape 214 des listes de masques.

A partir des listes de sorties SNOK et SOK, on construit deux listes de masques dont les adresses des premiers éléments sont contenues dans les pointeurs : masque_NOK et masque_OK.

Un élément de ces listes est une structure dont le type (MASQUE) est prédéfini dans l'en-tête « structures.h ». Le champ « masque » de ces structures est un entier du type prédéfini (TYPEMASQUE).

Il sert à coder les sorties citées dans la liste de sorties et relatives à un même groupe. Ce groupe est mémorisé dans la même structure par le champ « groupe » qui est un entier ordinaire (int).

Le principe du codage des sorties dans un masque est identique à celui adopté pour les marqueurs. Le nombre d'éléments d'une liste de masques est donc le nombre de groupes nécessaires pour repérer les sorties citées dans la liste de sorties associée. La correspondance entre le numéro de sortie et le groupe est déjà mémorisée dans le tableau groupeSortie[ ] cité précédemment.

Les champs « masques » de ces listes de masques sont destinés à être comparés au contenu des marqueurs du même groupe de chacun des nœuds pour la réalisation des opérations d'intersection et d'exclusion.

Pour que le mécanisme de marquage fonctionne, on ne doit pas avoir de référence multiple à une même sortie dans les liste de sorties.

Pour la construction de la liste masque_NOK, il ne faut pas tenir compte de la sortie choisie comme référence puisque c'est dans son cône que sera recherchée l'intersection.

Les différentes opérations d'intersection et d'exclusion nécessaires à la construction du sous-ensemble minimal se font en comparant les marqueurs des nœuds *de l'arbre de référence* avec les masques construits à partir des listes SNOK et SOK.

A l'étape 216, on procède ensuite à l'intersection des listes des SNOK.

Pour décider si un nœud est inclus dans l'intersection des listes des SNOK, on compare, pour chaque élément de la liste « masqueSnok », la valeur du masque et le marqueur relatif au même groupe associé à ce nœud.

Si le nœud appartient à l'intersection, tous les bits égaux à un dans les masques doivent l'être dans les marqueurs correspondants pour tous les groupes de la liste « masqueSnok ».

La comparaison se fait en utilisant le ET logique.

Ceci consiste, pour chaque élément de la liste de masques, à tester l'exactitude de la condition suivante : valeur_marqueur(groupe) & valeur_masque == valeur_masque.

Sur la figure 15 est représenté, d'une part, l'arbre de référence noté 1502 et, d'autre part, un autre arbre 1504 associés à une sortie incorrecte.

L'intersection des deux arbres notée 1506 est entourée d'une ligne pointillée. On constate sur cette figure, que la structure de l'intersection est celle d'un ensemble de sous-arbres 1508 et 1510, dont les sommets 1512 et 1514 respectivement sont reliés par un arc orienté à au moins un nœud de chacun des arbres 1502 et 1504.

La structure de l'intersection est celle d'un ensemble de sous-arbres disjoints de l'arbre de référence. Si ces arbres n'étaient pas disjoints, il existerait dans l'arbre de référence au moins un cycle, ce qui est impossible, par construction.

Pour définir l'intersection 1506, il suffit donc de savoir énumérer les sommets 1512 et 1514 de ces arbres.

Pour trouver les sommets de ces arbres, on parcourt l'arbre de référence depuis son sommet avec une procédure standard. Dés qu'on trouve un nœud qui appartient à l'intersection, il s'agit d'un sommet ; on l'enregistre dans une liste et l'algorithme rebrousse chemin comme s'il s'agissait d'une terminaison.

Les coupures interrompent également la descente mais ne sont pas prises en compte en tant que sommet. S'il y a lieu, leur jumeau peut être pris comme sommet quand il est analysé.

La méthode exposée dans le paragraphe précédent consiste à rechercher l'ensemble des nœuds qui influencent simultanément toutes les sorties de la liste des SNOK.

Il peut donc se faire que l'intersection trouvée soit vide. Ceci est le cas si les sorties défaillantes ont des cônes disjoints ou bien si l'intersection est non connexe comme dans le cas de la figure 14.

Lorsque l'intersection trouvée est vide, le programme recherche à l'étape 218 les intersections minimales connexes.

Ces sous-ensembles sont les intersections de cônes qui ne contiennent plus aucune autre intersection.

Avant de rechercher ces sous-ensembles, il est nécessaire de reconstruire la liste (interne) des SNOK et celle des masques_NOK afin de tenir compte de la sortie qui avait été prise comme référence.

Ensuite, la recherche s'effectue en trois phases.

La première phase consiste à rechercher la liste de tous les sous-ensembles qui font intervenir au moins une sortie de la liste SNOK dans leur définition (liste primaire).

Un sous-ensemble est repéré par le marqueur du nœud courant de ce sous-ensemble.

Cette liste est obtenue en balayant la liste des nœuds. Pour chaque nœud, on fait le ET logique entre les composantes du marqueur et leur homologue (de même numéro de groupe) de la liste masque_NOK. Si au moins un des résultats de ces ET est non nul, le marqueur de ce nœud définit un sous-ensemble pertinent. On vérifie si ce sous-ensemble n'a pas déjà été trouvé avant de l'enregistrer.

A la fin de cette étape, la liste primaire contient donc la désignation de tous les sous-ensembles formés à partir des cônes des SNOK. Chacun des sous-ensembles y figure de manière unique.

Du point de vue de la programmation, cette liste est une liste de structures de type « listeTableauxMasques ». Le champ « masque » de ces structures est un pointeur sur le tableau d'entiers de type (TYPEMASQUE) contenant, pour chaque groupe, le résultat des opérations ET citées ci-dessus.

Le champ « inclusion » sert, dans la seconde phase, à indiquer si le sous-ensemble contient un autre sous-ensemble.

La seconde phase consiste à comparer chaque élément de la liste primaire avec les autres.

La comparaison consiste à tester, pour chaque groupe, les conditions suivantes :

(masque_du_sous-ensemble_A) & (masque_du_sous-ensemble_B) == (masque_du_sous-ensemble_A).

Si pour tous les groupes cette condition est vraie, le sous-ensemble B est inclus dans A et A doit être rejeté puisqu'il ne peut, dès lors, être une intersection minimale. Pour signifier ce rejet, on met à un le champ inclusion de la structure décrivant le sous-ensemble A.

A la fin de cette phase, les éléments de la liste dont le champ inclusion n'a pas été mis à un sont les intersections minimales connexes recherchées.

Si « n » est le nombre de sorties de la liste SNOK, le nombre total de comparaisons à effectuer est égal à n(n-1).

La troisième phase consiste à passer en revue tous les éléments de la liste primaire et, pour ceux dont le champ « inclusion » est nul, à construire, à partir du champ « masque », un fichier texte contenant la liste des noms de sorties relatifs à ce sous-ensemble.

Chaque fichier issu du fractionnement du fichier des SNOK initial porte un nom construit de la manière suivante : « nomListeInitialeSNOK_fraction_numéro.snok ». Le programme devra être relancé pour chacune de ces fractions de la liste initiale.

L'étape suivante, notée 220, consiste en l'exclusion des SOK et des entrées.

Il s'agit de retirer de l'intersection les nœuds qui appartiennent à l'USOK (réunion des cônes des SOK).

Dans le programme, on distingue les exclusions exhaustives et non exhaustives.

Pour la construction du sous-ensemble minimal, on utilise le mode « non exhaustif ».

Dans ce mode, on parcourt chacun des arbres de l'intersection et lorsqu'un nœud appartient à la réunion des SOK, on met à « un » le champ « exclusion » dans la structure NŒUD qui lui est attachée.

Le parcours se fait avec l'algorithme « à ordre préfixé » les coupures sont des points de rebroussement et leur jumeau peut être exclu s'il appartient à l'USOK.

Pour déterminer si un nœud appartient à l'USOK, pour chaque élément de la liste masque_OK, on compare la valeur du masque avec

l'élément (de même numéro de groupe) du tableau de marqueurs associé à ce nœud. La comparaison utilise le ET logique. Si pour l'une au moins de ces comparaisons les deux entiers ont un bit homologue égal à un, le nœud appartient à l'USOK. Il s'agit donc de tester l'exactitude des conditions suivantes : valeur_marqueur(groupe) & valeur_masque != 0.

L'exclusion des nœuds connectés à la sortie des cellules d'entrée se fait, de façon triviale, en balayant la liste des broches du composant.

A l'étape 222, est défini un sommet virtuel.

Les sommets du sous-ensemble minimal sont reliés à un nœud virtuel.

La variable globale « sommet_virtuel » est un pointeur sur ce nœud.

La fonction « **créer_un_pseudo_sommet( )** » crée ce nœud et initialise ses champs.

La numérotation du sous-ensemble minimal, réalisée à l'étape suivante notée 224, consiste à mémoriser dans le champ « compteur_ancêtres » de chaque nœud le nombre total de ses ancêtres.

On attribue aux nœuds exclus et aux coupures un nombre d'ancêtres nul.

Un nœud exclu offre une contribution nulle à son successeur.

Une coupure offre une contribution d'une unité à son successeur.

Les parents d'un nœud sont ses ancêtres immédiats (directement rattachés).

La figure 16 donne un exemple de numérotation.

Cette numérotation est utilisée dans l'algorithme de localisation pour proposer des choix de nœud à tester.

Pour effectuer la numérotation d'un arbre, on parcourt une première fois cet arbre avec l'algorithme standard à ordre préfixé et on attribue aux champs « compteur_ancêtres » la valeur nulle (phase d'initialisation). Dans une seconde phase, on refait le parcours avec l'algorithme à ordre post-fixé et on attribue au compteur d'ancêtres de chaque nœud la valeur :

(somme des compteurs d'ancêtres des parents de ce nœud + nombre de parents du nœud)

La phase suivante du procédé désignée par la référence générale 124 sur la figure 1 consiste en la recherche des fautes par dichotomie dans le sous-arbre minimal.

A cet effet, on détermine d'abord, à l'étape 230, suivant un critère qui sera explicité dans la suite, un nœud moyen dans le sous-arbre minimal puis on effectue à l'étape 232 un test physique sur ce nœud si ce nœud est testable. Pour évaluer si le nœud est testable, une vérification est effectuée à l'étape 233. Si la réponse est négative, plutôt que de solliciter le testeur en 232, un appel à la phase 126 du procédé est effectué.

L'algorithme de dichotomie assure la convergence de la recherche vers une ou plusieurs zones défaillantes. Le principe simplifié est le suivant :

L'algorithme propose un test sur un nœud interne du domaine de recherche.

Ce nœud est choisi de telle sorte que le nombre de ses ancêtres soit approximativement égal à la valeur moyenne du nombre d'ancêtres des nœuds du domaine de recherche, non encore testés.

Si le signal sur ce nœud est normal, ce nœud est éliminé de la recherche ainsi que tous ses ancêtres.

Ceci est basé sur l'hypothèse selon laquelle si un nœud n'est pas un point de propagation de faute, ses ancêtres, qui sont les nœuds susceptibles de l'influencer ne le sont pas non plus.

On obtient donc ainsi une réduction, à caractère exponentiel, de la zone de recherche.

Si le signal sur le nœud testé est anormal, c'est que ce nœud appartient à une zone défaillante. Il est alors pris comme nouveau point de départ des recherches.

Comme certains nœuds ne sont pas testables et que les nœuds trouvés mauvais ne sont pas forcément des sommets de zones défaillantes, des tests de cohérence sont nécessaires avant de pouvoir déterminer les limites d'une zone défaillante. La présence de ces tests de cohérence, imbriqués dans la dichotomie, complique singulièrement l'algorithme. Celui-ci est décrit en détail dans la suite de la description.

Le sous-ensemble minimal dans lequel les fautes sont recherchées est un ensemble d'arbres disjoints. Les sommets de ces arbres forment la liste des parents d'un nœud virtuel destiné à manipuler le sous-ensemble minimal de façon globale.

En fonction du moyen de test utilisé, on spécifie une profondeur limite de testabilité des nœuds (profondeur_maxi).

Suivant l'implantation du circuit (désignée par "layout" en anglais), la profondeur limite spécifiée peut rendre une partie des nœuds non-testables (y compris les parents du nœud virtuel).

On peut, de plus, déclarer une partie des nœuds non-testables indépendamment de leur profondeur.

Les terminaisons, testables ou non, du sous-ensemble minimal peuvent être :

- Des coupures ;
- Des nœuds de sortie de cellules d'entrée (déjà exclus) ;
- Des nœuds frontières d'USOK (déjà exclus).

A chaque nœud est affectée une variable contenant le nombre total de ses ancêtres. Les coupures et les nœuds d'entrée ont un nombre d'ancêtres nul.

Les caractères du nœud virtuel sont, par convention :

- Non-testable
- Défaillant
- Non exclu

En outre, les données suivantes sont également mises en œuvre par le programme :

- La liste des nœuds internes déjà testés (liste facultative). Cette liste est complétée automatiquement par le programme au fur et à mesure que des nouveaux tests sont effectués. Si elle est inexistante, le programme la crée.
- Une liste (facultative) de nœuds que l'opérateur, pour des raisons diverses, considère comme non-testables.
- Le fichier donnant le "niveau métal" de chacun des nœuds internes à tester (facultatif). Tout nœud non cité dans cette liste est considéré, par

défaut, comme étant au niveau "métal 1". Si la liste n'existe pas, tous les nœuds sont considérés comme étant au niveau "métal 1".

En effet, un nœud donné est toujours situé sur une "piste" d'interconnexion. En général, les pistes d'interconnexion sont réparties sur des "niveaux" situés à différentes profondeurs. Ces niveaux d'interconnexion sont séparés par des couches isolantes (SiO2, Si3N4, etc. ...).

Le premier niveau d'interconnexion appelé "métal 1" est, par convention, celui qui est le plus proche de la surface. Le dernier est celui qui est le plus proche du semi-conducteur. Le programme supporte 253 niveaux mais on pourrait étendre ce nombre sans problème.

Mesurer l'état d'un nœud à partir d'un testeur suppose que ce testeur sache sonder le circuit jusqu'à la profondeur d'interconnexion associée à ce nœud. Pour un testeur donné, il existe donc une limite de visibilité des nœuds. Cette limite de testabilité est appelée "profondeur maximale de test" (variable "profondeur_maxi" dans le code). Elle doit être spécifiée par l'utilisateur suivant le moyen de test utilisé.

Les fichiers, facultatifs, dont on dispose sont :

- Un fichier donnant la profondeur des nœuds (niveau métal = 1 par défaut)
- Un fichier de nœuds non-testables (seuls les nœuds spécifiés sont non-testables)
- Un fichier de nœuds déjà testés (créé par le programme s'il est inexistant)

Les paramètres de recherche sont :

- La profondeur maximale de test
- Le critère d'arrêt des recherches (une cellule défaillante ou une zone défaillante ou toutes les parties défaillantes).

L'algorithme de recherche est basé sur le principe de dichotomie. Il assure, généralement, la convergence de la recherche vers une ou plusieurs zones défaillantes.

L'algorithme propose un test sur un nœud interne du domaine de recherche. Ce nœud est choisi de telle sorte que le nombre de ses ancêtres soit approximativement égal à la valeur moyenne du nombre d'ancêtres des

noeuds du domaine de recherche, non encore testés. Le nœud choisi est dit "nœud moyen".

Ce nœud doit, de plus, satisfaire les contraintes suivantes :

- Il doit être testable ;
- Ne pas avoir déjà été trouvé défaillant
- Ne doit pas être un sommet de zone défaillante
- Ne doit pas être exclu
- Ne doit pas être l'ancêtre d'un nœud exclu après test
- Ne doit pas être l'ancêtre d'un sommet de zone défaillante

La réponse concernant le résultat du test du signal sur ce nœud peut être fournie automatiquement par le testeur ou bien entrée au clavier par l'opérateur.

Si le signal sur ce nœud est normal, ce nœud est éliminé de la recherche ainsi que tous ses ancêtres.

Pour éliminer ce nœud, on l'exclut, ainsi que tous ses ancêtres, en mettant à un les champs « exclusion » des structures de données associées à ces nœuds (exclusion dite exhaustive). Une fois cette exclusion opérée, on renumérote (en nombre d'ancêtres) le sous-ensemble minimal depuis le sommet virtuel, de telle sorte que le nouveau calcul de nœud moyen ne tienne plus compte des nouveaux nœuds exclus.

Si le signal sur le nœud testé est anormal, c'est que ce nœud est en aval d'une zone défaillante. Il est alors pris comme nouveau point de départ des recherches.

La technique de programmation adoptée est récursive. La même fonction est utilisée avec certains paramètres d'entrée hérités du contexte précédent. Notamment, le sommet de la recherche est maintenant le nœud dont le signal a été déclaré mauvais.

On n'effectue pas de récurrence sur une coupure. On se contente de la marquer mauvaise si le signal est mauvais sur son nœud jumeau.

Au retour de chaque récurrence, suivant les décisions prises relativement au critère d'arrêt spécifié, on peut sortir de la fonction ou bien continuer à chercher parmi les nœuds proposables non encore testés.

Lorsque la recherche doit se poursuivre, c'est-à-dire que l'on souhaite trouver toutes les zones défaillantes ou bien que les nœuds testés mauvais et les tests de cohérence effectués ne permettent pas, à ce stade, de déclarer une zone défaillante.

Des tests de cohérence sont nécessaires avant de pouvoir déterminer les limites d'une zone défaillante.

Ceci est lié au fait que certains nœuds ne sont pas testables et que les nœuds trouvés mauvais ne sont pas forcément des sommets de zones défaillantes.

Les tests de cohérence et les tests d'exactitude du critère d'arrêt spécifié sont effectués lors de la phase 126 d'"analyse après épuisement".

Le premier appel de l'algorithme de recherche est fait depuis le sommet virtuel du sous-ensemble minimal. La zone de recherche initiale est donc le sous-ensemble minimal.

L'algorithme de recherche constituant la fonction **dichotomie( )** est explicité dans l'organigramme de la figure 17.

La fonction **dichotomie ()** reçoit en entrée le critère d'arrêt 1702, le sommet virtuel 1704 de l'arbre sur lequel elle opère ainsi que le sommet 1706 de la recherche.

A l'étape 1708, un nœud moyen satisfaisant les contraintes spécifiées est proposé. Ce nœud moyen est déterminé comme cela sera expliqué ultérieurement par une mise en œuvre de la fonction **chercher_nœud_moyen ()** exposée en regard de la figure 18.

Il est vérifié à l'étape 1710 que ce nœud moyen existe. Si tel n'est pas le cas, l'étape 1712 est mise en œuvre au cours de laquelle le reste de l'arbre est analysé sans mise en œuvre du testeur. A cet effet, la fonction **analyser_cellules_restantes ()** est mise en œuvre pour tenter de fractionner l'ensemble des nœuds restant en zone défaillante. La fonction **analyser_cellules_restantes ()** sera décrite ultérieurement en regard de la figure 23 en tenant compte des algorithmes qu'elle met en œuvre et qui sont décrits aux figures 24 à 28.

Si par contre, il est révélé à l'étape 1710 qu'un nœud moyen satisfaisant les contraintes spécifiques existe, ce nœud est testé à l'étape 1714 par mise en œuvre du testeur.

A cet effet, une séquence de tests prédéterminés est appliquée aux bornes d'entrée du circuit intégré et le signal obtenu au point de mesure correspondant au nœud moyen déterminé à l'étape 1708 est mesuré et comparé au signal théorique devant être obtenu en ce point.

Si le signal mesuré est identique au signal théorique devant être obtenu, le nœud est considéré comme bon. Sinon, le nœud est considéré comme mauvais.

Suite aux tests opérés à l'étape 1716, si le nœud est bon, celui-ci est exclu à l'étape 1718 et l'arbre de recherche est numéroté depuis le sommet virtuel défini en 1704. L'étape 1708, ainsi que les étapes ultérieures sont remises en œuvre sur la seule partie de l'arbre obtenue après exclusion du nœud moyen précédemment testé.

Si par contre à l'étape 1716, le nœud est détecté comme mauvais, celui-ci est marqué comme mauvais à l'étape 1720. L'étape ultérieure 1722 consiste à vérifier si le nœud marqué mauvais est ou non une coupure. Si tel est le cas, un autre nœud de l'arbre est testé par une nouvelle mise en œuvre des étapes 1708 et suivantes.

Par contre, si le nœud marqué mauvais n'est pas une coupure, ce nœud devient à l'étape 1724 le nouveau sommet de recherche servant à une nouvelle mise en œuvre récursive de la fonction **dichotomie** () appliquée avec comme nouveau sommet de recherche en 1706 le nœud moyen précédemment considéré comme mauvais.

A l'issue de l'application récursive de la fonction **dichotomie** () à l'étape 1724, il est testé à l'étape 1726 si le critère d'arrêt considéré en 1702 est ou non satisfait. Tant que ce critère n'est pas satisfait, de nouveaux nœuds moyens sont proposés à l'étape 1708 et testés aux étapes ultérieures. Dès que le critère d'arrêt est satisfait à l'étape 1726, il est mis un terme à l'étape 1728 à la fonction dichotomie et le résultat de l'analyse est mise à la disposition de l'utilisateur.

La recherche du nœud moyen opérée à l'étape 230 de la figure 2 correspondant à l'étape 1708 de la figure 17 s'effectue en deux phases.

Dans la première phase notée 234, on parcourt l'arborescence dans laquelle s'effectue la recherche et on calcule la valeur moyenne du nombre d'ancêtres.

Dans la seconde phase notée 236, on reparcourt l'arborescence et on cherche le nœud dont le nombre d'ancêtres est le plus proche, inférieurement, de la valeur moyenne calculée à l'issue de la première phase.

Un indicateur de coupure du nœud moyen proposé est également donné.

La fonction **chercher_noeud_moyen ( )** qui assure la coordination de ces deux phases est décrite en regard de la figure 18 dans l'organigramme du calcul et de la localisation du nœud moyen. Les fonctions relatives aux deux phases sont décrites dans les paragraphes suivants, respectivement au regard de la figure 19 et des figures 19, 20 et 21.

La fonction **chercher_nœud_moyen ()** mise en œuvre à l'étape 1708, dispose en entrée, en 1802, du sommet de la recherche, c'est-à-dire du sommet de l'arbre dans lequel la recherche du nœud moyen est effectuée. A l'étape 1804, un test est effectué pour déterminer si ce sommet est NULL, c'est-à-dire si l'arbre est vide. Si tel est le cas, à l'étape 1806, la variable solution est fixée à 0, ainsi que la variable indic_coupure. Le résultat de la fonction est retourné à l'étape 1808 avec les valeurs associées à solution et indic_coupure. Les définitions de solution et indic_coupure seront données dans la suite de la description en référence aux figures 20 et 21.

En revanche, si l'arbre n'est pas vide, c'est-à-dire que le sommet testé à l'étape 1804 est non nul, l'étape 1810 est mise en œuvre par application de la fonction **calcul_moyenne ()** qui sera décrite en référence à la figure 19. Lors de cette étape, le nombre total de nœuds de l'arbre est calculé par à un parcours de celui-ci. De même, le calcul du cumul des nombres d'ancêtres est effectué. Le résultat de ces calculs est mémorisé respectivement dans les variables nombre_total_de_nœuds et cumul_des_nombres_d'_ancêtres.

Lors du test de l'étape 1812, le nombre_total_de_nœuds est comparé par la valeur 1. Si le nombre_total_de_nœuds est égal à 1, la moyenne est prise égale au cumul_des_nombres_d'_ancêtres à l'étape 1814.

En revanche, si le nombre_total_de_nœuds est supérieur à 1, la moyenne est calculée à l'étape 1816.

Quel que soit le mode de calcul de la moyenne à l'étape 1814 ou 1816, l'algorithme procède à l'étape 1818 à la localisation du nœud moyen, permettant de déterminer la variable solution ainsi que l'indicateur de coupure noté indic_coupure. L'algorithme de localisation du nœud moyen sera décrit ultérieurement en référence à la figure 19 modifiée conformément aux figures 20 et 21.

A l'étape ultérieure 1820, la solution obtenue pour le nœud moyen à l'étape 1818 est comparée aux valeurs particulières NULL et sommets. Si la variable solution est égale à l'une de ces valeurs, les variables solution et indic_coupure sont fixées à 0 à l'étape 1822. Dans tous les cas, l'étape 1808 est enfin mise en œuvre mettant à disposition le résultat de la fonction **recherche_nœud_moyen ()**.

La fonction **calcul_moyenne ()** décrite sur la figure 19 dispose en entrée du sommet 1902 de l'arbre considéré, du total des ancêtres de cet arbre 1904, ainsi que nombre de nœuds 1906 de l'arbre considéré.

A l'étape 1908, il est vérifié que le sommet considéré en 1902 est testable. Si tel est le cas, à l'étape 1910, la variable nombre_nœuds est incrémentée et le nombre d'ancêtres du sommet est cumulé dans la variable total_ancêtres.

Que le sommet considéré en 1902 soit ou non testable, l'étape ultérieure 1912 consiste à proposer un parent du sommet, ceci afin de parcourir progressivement l'arbre. Si lors du test effectué à l'étape 1914, il est constaté qu'il n'existe pas de parent au sommet considéré, il est mis un terme à la fonction **calcul_moyenne ()** à l'étape 1916 et les résultats de la fonction à savoir la variable nombre_nœuds et l'accumulateur sont retournés.

En revanche, si lors du test de l'étape 1914, le parent proposé pour le sommet existe, il est vérifié à l'étape 1918 que ce parent est exclu ou mauvais. Si tel est le cas, l'étape 1912 est à nouveau mise en œuvre.

Si tel n'est pas le cas, il est testé à l'étape 1920 que le parent considéré est ou non une coupure. S'il ne s'agit pas d'une coupure, la fonction **calcul_moyenne ()** est mise en œuvre de manière récursive à l'étape 1922. A l'issue de cette mise en œuvre récursive, l'étape 1912 de proposition d'un autre parent du sommet considéré est à nouveau exécutée.

En revanche, si à l'étape 1920, il est conclu que le parent considéré est une coupure, il est vérifié à l'étape 1924 si cette coupure est ou non testable. Si elle n'est pas testable, un nouveau parent est proposé à l'étape 1912. Si cette coupure est testable, la variable nombre_nœuds est incrémentée à l'étape 1926, avant qu'un nouveau parent du sommet ne soit proposé à l'étape 1912.

Pour le calcul de la moyenne à l'étape 234 de la figure 2 correspondant à l'étape 1810 de la figure 18, on parcourt l'arbre et, pour chaque nœud pertinent, on cumule dans un accumulateur le nombre de ses ancêtres et on incrémente un compteur de nœuds.

On ignore les nœuds exclus ou testés mauvais ainsi que leurs ancêtres. La récurrence se produit sur les nœuds pertinents qui ne sont pas des coupures. Les coupures non testables sont ignorées. Les coupures testables ne donnent lieu qu'à l'incrémentation du compteur de nœuds (nombre d'ancêtres nul).

Pour le premier appel de la fonction, on initialise à zéro l'accumulateur et le compteur de nœuds.

Pour le premier appel, le sommet donné ne doit pas être un nœud exclu.

L'algorithme détaillé du calcul du nombre de nœuds et du cumul des ancêtres constituant la fonction **calcul_moyenne()** est donné dans l'organigramme de la figure 19.

La procédure de localisation du nœud moyen à l'étape 236 de la figure 2 correspondant à l'étape 1818 de la figure 18 est semblable à la procédure précédente moyennant quelques adaptations.

Le nom de cette procédure est : **localiser_moyenne( )**.

Pour en obtenir l'organigramme de la fonction **localiser_moyenne ()**, il suffit de remplacer l'étape 1910 par le bloc de la figure 20, et l'étape 1926 par le bloc de la figure 21.

D'autre part, les paramètres d'entrée sont : Sommet, moyenne, Solution, indic_coupure et « precedent ». Le paramètre « precedent » est la dernière meilleure valeur approchée trouvée pour le nombre d'ancêtres du nœud moyen. Le premier appel de la fonction se fait avec precedent = 0.

Solution est le meilleur nœud trouvé et indic_coupure son indicateur de coupure.

Les nœuds déjà déclarés comme sommets de zones défaillante ne sont pas proposés comme nœuds moyens puisqu'ils sont interprétés comme nœud mauvais et écartés.

Plus précisément, et comme représenté sur la figure 20, lorsqu'à l'étape 1908, le sommet est considéré comme testable, il est calculé une variable logique T à l'étape 2002. Celle-ci est donnée par :

T = (Sommet->compteur_ancetres=<moyenne)

&&

(Sommet-compteur_ancetres>=precedent).

Un test de la valeur de la variable T est effectué à l'étape 2004. Si cette variable est fausse, un nouveau parent du sommet est proposé à l'étape 1912.

En revanche, si la variable T est vraie, alors à l'étape 2006, les valeurs des variables précédentes indic_coupure_sommet sont modifiées comme indiqué ci-dessous :

precedent=Sommet->compteur_ancetres

Indic_coupure=0

Solution=Sommet

De même, en remplacement de l'étape 1926, et comme représenté sur la figure 21, il est d'abord défini à l'étape 2102 une variable logique T. Celle-ci est définie par :

T= (nœud->compteur_ancetres=<moyenne)

&&

(nœud->compteur_ancetres>=precedent)

A l'étape 2104, la valeur logique de la variable T est testée. Si celle-ci est fausse, un nouveau parent du sommet est proposé à l'étape 1912.

En revanche, si la variable T est vraie, à l'étape 2106, les valeurs des variables précédentes, indic_coupure et solution sont modifiées comme indiqué ci-dessous :

Precedent=nœud->compteur_ancetres

Indic_coupure=1

Solution=noeud

Après détermination du nœud moyen, le test de celui-ci se fait par sollicitation du testeur à l'étape 232 de la figure 2, correspondant à l'étape 1714 de la figure 17 avec la fonction **tester_nœud( )**. Cette fonction a comme paramètre d'appel l'adresse du nœud à tester.

Avant de solliciter le testeur, cette fonction vérifie si le nœud n'a pas déjà été testé en analysant le contenu du fichier des nœuds testés. Si le résultat est connu, il est immédiatement utilisé dans le programme. Dans le cas contraire, cette fonction interroge le testeur sur ce nœud. La réponse du testeur, indépendamment de son utilisation dans le programme, est archivée dans le fichier des nœuds testés. Ceci évite, si on doit relancer le programme, d'avoir à refaire des tests sur ces nœuds.

La réponse fournie si le nœud a un fonctionnement correct est « 1 ». Si le nœud est trouvé défaillant, la réponse doit être « 0 ».

La sollicitation du testeur est assurée par la fonction **interroger_testeur( )**. Cette fonction écrit dans un fichier texte le nom du nœud à tester et vient périodiquement relire le contenu de ce fichier. Lorsque le testeur a répondu, la fonction retourne la réponse à la fonction **tester_nœud( )**.

Le nom du fichier d'interface (absolu ou relatif) est défini dans le fichier d'en-tête « general.h ». Dans ce fichier sont également définis les autres paramètres intervenant dans le contrôle du testeur (durée entre deux lectures du fichier d'interface et dépassement de temps sur la réponse testeur).

Les opérations d'écriture et de lecture dans le fichier d'interface ne nécessitent aucune synchronisation entre le testeur et le programme.

Des contrôles de cohérence sur le nom du nœud et sur la réponse du testeur sont effectués dans la fonction **interroger_testeur( )** à chaque test.

Le dialogue entre le testeur et le programme suit le protocole suivant :

- Le programme vide ou crée le fichier d'interface et écrit sur la première ligne le nom du nœud à tester sans faire de retour à la ligne.
- Le testeur est censé venir lire périodiquement ce fichier. Dès qu'il détecte la présence d'un nom de nœud nouveau, il effectue le test et répond (en mode « append ») par un retour à la ligne suivi de l'entier 0 ou 1 non suivi d'un retour à la ligne.
- Le programme vient lire périodiquement le fichier d'interface. Dès qu'il détecte la ligne supplémentaire, il sait que le testeur a répondu et le contenu de cette ligne est interprété numériquement par la fonction standard « **fscanf( )** » du langage C.

Les flux du fichier d'interface vers le programme et le testeur sont distincts et peuvent être simultanés.

Un programme « pseudo_testeur » peut être lancé en arrière plan (option & du shell). Il permet d'écrire manuellement dans le fichier d'interface. L'opérateur peut ainsi répondre au clavier à la place du testeur. L'accès simultané au fichier d'interface est ainsi possible par trois flux (le programme, le testeur et le pseudo_testeur).

Pour écrire le programme d'accès du testeur au fichier d'interface, on peut partir du fichier source « pseudo_testeur.c » et des en-têtes associées.

L'algorithme de dichotomie est mis en œuvre jusqu'à épuisement du sous-arbre minimal. L'analyse après épuisement d'un niveau objet de la phase 126 dans l'organigramme de la figure 1 est demandée par l'algorithme de dichotomie quand il ne reste plus de nœud proposable à tester.

Il s'agit alors de tenter de fractionner l'ensemble des nœuds restant en zones défaillantes.

Suivant le critère d'arrêt choisi et le résultat des tentatives de fractionnement, on restitue soit la valeur 0 (critère non satisfait) soit la valeur 1 (critère satisfait).

Pour un contexte donné de l'algorithme de dichotomie, lorsque tous les nœuds proposables ont été épuisés, les extrémités du sous-arbre analysé ne peuvent être que :

- Des nœuds de sortie de cellules d'entrée. Ces nœuds sont déjà exclus dés le départ. Ils seront dénommés « nœuds d'entrée » ;
- Des nœuds frontières d'USOK (déjà exclus dés le départ) ;
- Des coupures exclues ;
- Des coupures testées mauvaises ;
- Des coupures non testables.

Les nœuds internes restant peuvent être :

- Des nœuds non testables ;
- Des nœuds déclarés « sommet de zone défaillante » ;
- Des nœuds testés mauvais mais non déclarés « sommet de zone défaillante ».

Remarques :

- Il ne peut pas exister d'extrémité non testable qui ne soit ni une entrée, ni une frontière d'USOK, ni une coupure.
- Le sommet de l'arborescence à analyser ne peut être qu'un nœud testable et trouvé mauvais (il ne peut s'agir d'une coupure).
- On considère que les ancêtres d'un sommet de zone défaillante ne sont plus accédés par la suite.

Pour décider si une zone est défaillante, on se réfère à la définition donnée en regard de la figure 5. Il découle de cette définition que la présence de coupures testées mauvaises ou de coupures non testables constitue un obstacle pour déclarer une zone défaillante. Il est donc nécessaire d'analyser l'environnement de ces coupures dans le reste de l'arbre de manière à lever le doute toutes les fois où cela est possible. Cette opération est dénommée "résolution des coupures". Elle consiste à rechercher si les coupures présentes ont leur nœud jumeau dans la zone de recherche en cours.

Sur la figure 22 est donné un exemple de résolutions autorisées et interdite, un exemple "d'effet gigogne" et un exemple de trois zones défaillantes séparables. "L'effet gigogne" sera défini dans la suite de la description.

Les nœuds de l'arbre représentés sur la figure 22 sont différenciés en reprenant les conventions graphiques utilisées jusqu'alors. Ainsi, les nœuds mauvais sont désignés par un rond au centre duquel figure croix. Les nœuds non testables sont désignés par un rond à l'intérieur duquel figure un point d'interrogation, les nœuds bons sont désignés par un rond à l'intérieur duquel figure un autre rond, et les coupures sont désignées par un rond à l'intérieur duquel figure un carré.

Les coupures constituant des nœuds non testables ou des nœuds mauvais présentent intérieurement outre un carré, respectivement une croix et un point d'interrogation.

Sur l'arbre de la figure 22, trois zones défaillantes désignées par les références 2202, 2204 et 2206 sont entourées par un cerclage en pointillés. Ces zones défaillantes présentent par définition un sommet constitué d'un nœud mauvais, qui pour des raisons de clarté a été encadré par un rectangle. Ces nœuds mauvais ont pour parent au moins un nœud non testable et éventuellement des nœuds bons.

Les trois zones défaillantes sont séparées puisqu'elles ne comportent pas de nœud commun.

A l'intérieur de la zone défaillante 2206, figurent deux coupures 2208 et 2210. Ces coupures correspondent à des nœuds jumeaux respectivement des nœuds 2212 et 2214. L'association des coupures avec leurs nœuds jumeaux est représentée par un lien pointillé 2216 et 2218.

Dans la mesure où les coupures 2208 et 2210 ont leurs nœuds jumeaux dans une même limite de zone défaillante 2208, la résolution est autorisée.

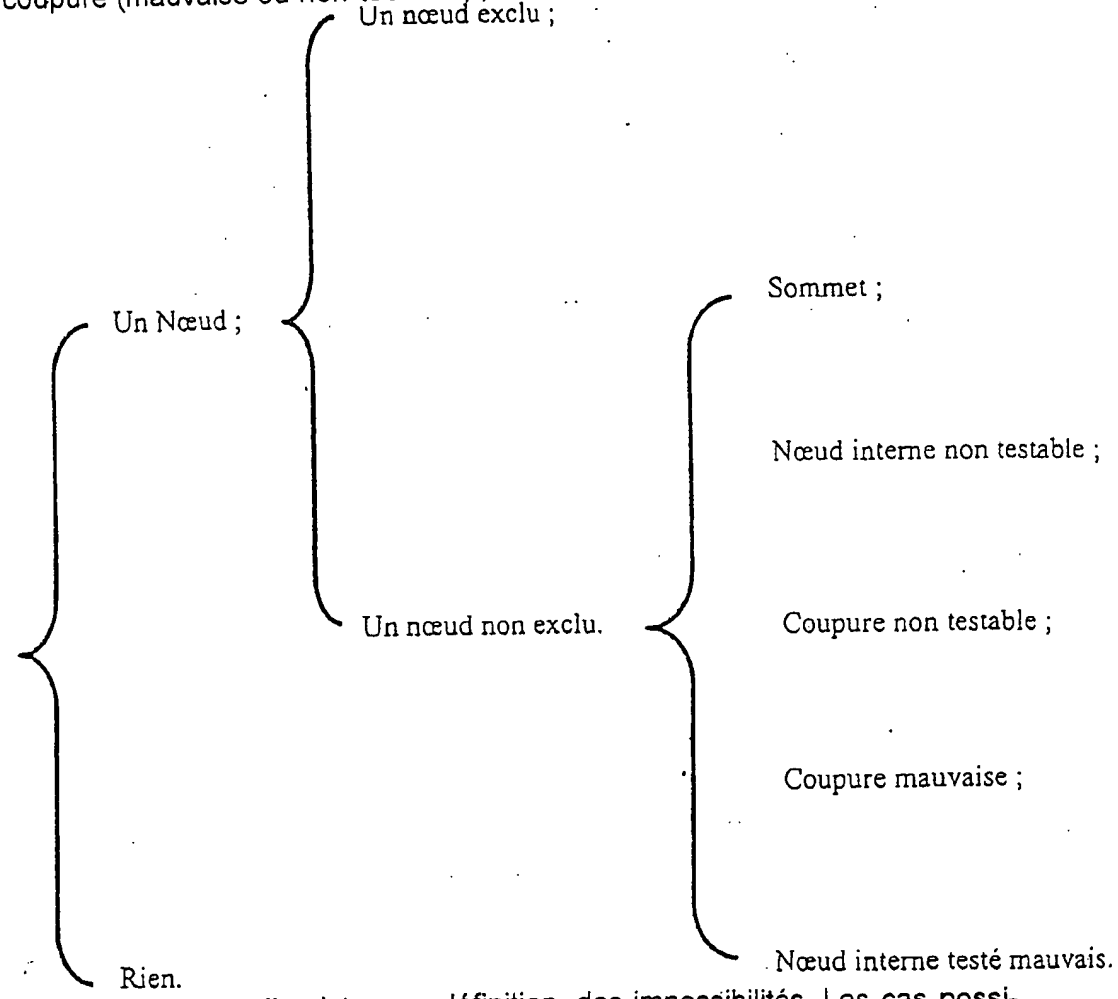
En revanche, la coupure notée 2218 ayant pour nœud jumeau le nœud 2220 comme indiqué par le lien pointillé 2222 se situe hors de la limite de zone défaillante 2206.

Il est donc inutile de tenter de résoudre cette coupure par accès interne à la zone déclarée défaillante 2206 étant donné que la zone défaillante qui pourrait en résulter inclurait de toute façon la première sans apporter, apparemment, d'information supplémentaire.

D'autre part, une zone ne doit pas être déclarée défaillante si au moins une de ses terminaisons est un sommet de zone déjà déclarée défaillante. On appelle ce phénomène « l'effet gigogne ». Les algorithmes décrits ci-après sont adaptés pour le détecter et en tenir compte.

Des exemples d'effets gigognes identifiés par les références 2230 et 2234 apparaissent sur la figure 22 aux sommets des zones défaillantes 2202 et 2206.

Dans le sous-arbre où est faite l'analyse des nœuds restant, une coupure (mauvaise ou non-testable) peut être connectée à :



Cependant, il existe, par définition, des impossibilités. Les cas possibles et impossibles sont résumés dans le tableau ci-dessous :

Les impossibilités de résolution sont repérées par la lettre "I" ; elles sont dues à des conflits "bon / mauvais" ou "testable / non-testable". La lettre "P" indique que la résolution de la connexion est possible.

		Coupure mauvaise	Coupure Non testable
Connectée à :	Nœud exclu	I	P
	Sommet	P	I
	Nœud interne non testable	I	P
	Coupure non testable	I	P
	Coupure mauvaise	P	I
	Rien	P	P
	Nœud interne mauvais	P	I

Ceci conduit au tableau suivant résumant les possibilités de résolution des coupures. Les lettres "R" indiquent que la coupure peut être résolue ; la lettre "D" indique que le doute ne peut pas être levé.

Les cases vides correspondent à des résolutions impossibles.

		Coupure mauvaise	Coupure Non testable
Connectée à :	Nœud exclu		R
	Sommet	R	
	Nœud interne non testable		R
	Coupure non testable		D
	Coupure mauvaise	D	
	Rien	D	D
	Nœud interne mauvais	R	

En résumé, le critère de résolution peut s'énoncer :

Une **coupure testée mauvaise** sera résolue si elle a pour jumeau (dans la zone à analyser) le **sommet** ou un **nœud interne testé mauvais**.

Une **coupure non-testable** sera résolue si elle a pour jumeau dans la zone à analyser) un **nœud exclu** (frontière d'USOK ou nœud d'entrée) ou un **nœud interne non-testable**.

L'analyse des nœuds restants consiste donc à dresser la liste des candidats pouvant être déclarés sommet de zone défaillante. Pour chacun

de ces candidats, on analyse le sous-arbre dont il est le sommet en appliquant le critère de résolution des coupures. La résolution de toutes les coupures sous ce nœud est la condition nécessaire mais non suffisante pour que la zone puisse être déclarée zone défaillante.

L'algorithme détaillé de l'analyse après épuisement constituant la fonction **analyser_cellules_restantes( )** est donné sur l'organigramme de la figure 23.

La fonction **analyser_ cellules_restantes ()** dispose en entrée du sommet 2302 de l'arbre considéré.

La valeur de ce sommet est testée à l'étape 2304. Si ce sommet est NULL, la variable réponse est fixée à 0 à l'étape 2306 et la valeur de la variable réponse est retournée à l'étape 2308 en tant que résultat de la fonction **analyser_cellules_restantes ()**.

En revanche, si le sommet est différent de NULL à l'étape 2304, la liste des candidats à la résolution est construite à l'étape 2310 par mise en œuvre de la fonction **construire_liste_candidats ()** qui sera décrite en regard de la figure 24.

Lors du test effectué à l'étape 2312, si la liste de candidats est nulle, alors la valeur de la variable réponse est fixée à 0 à l'étape 2314 puis l'étape 2308 est mise en œuvre.

En revanche, si la liste est non nulle, la fin de la liste est considérée à l'étape 2316 puis à l'étape 2318, un élément de la liste est choisi en remontant celle-ci.

Si lors du test effectué à l'étape 2320, on constate qu'aucun élément ne peut être choisi, la liste est libérée à l'étape 2322 et la variable réponse est fixée à 0 à l'étape 2324 avant que l'étape 2308 ne soit effectuée.

En revanche, si un élément de la liste peut être choisi, il est procédé, à l'étape 2326, à l'analyse des nœuds situés sous ce nœud considéré alors comme au sommet. Cette analyse est effectuée par mise en œuvre de la fonction **analyser_sous_ce_sommet ()** qui sera décrite ultérieurement en référence à la figure 25.

Après l'analyse sous le nœud choisi, il est vérifié à l'étape 2328 si le critère est ou non satisfait. Si tel n'est pas le cas, les étapes 2318 et sui-

vantes sont à nouveau mises en œuvre. Au contraire, si le critère est satisfait à l'étape 2330, la variable réponse est fixée à 1 avant que cette réponse soit retournée à l'étape 2308.

Lorsqu'on est dans un contexte donné des récurrences de l'algorithme de dichotomie et qu'on exécute la phase d'analyse du reste de l'arbre, il s'agit de trouver d'éventuelles zones pouvant être déclarées défaillantes. Les différents candidats pouvant tenir le rôle de sommet pour ces zones sont les nœuds testés mauvais et non encore déclarés sommets de zone défaillante.

Pour avoir des zones défaillantes de taille minimale, on doit rechercher les sommets de ces zones parmi les nœuds testés mauvais les plus proches des entrées et remonter ensuite vers le sommet.

Dans un premier temps, on construit à l'étape 240 de la figure 2 correspondant à l'étape 2310 de figure 23 la liste de ces candidats. Pour cela, on parcourt l'arbre suivant la procédure à ordre « post-fixé » en ignorant les nœuds exclus, les coupures et les sommets de zones défaillantes ainsi que leurs ancêtres. On n'enregistre que les sommets testés mauvais que l'on rencontre.

La liste construite doit être relue de la fin vers le début par utilisation du pointeur de parcours inverse.

La liste comporte en fait au moins un élément à savoir : le sommet.

L'algorithme détaillé de construction de la liste constituant la fonction **construire_liste_candidats( )** est donné dans l'organigramme de la figure 24.

La fonction **construire_liste_candidats ()** reçoit en entrée le sommet 2402 de l'arbre considéré. A l'étape 2404 un parent du sommet est proposé. Il est vérifié lors du test de l'étape 2406 si un tel parent existe. Si ce parent existe, il est défini à l'étape 2408 une variable logique notée condition. Cette variable est définie par :

Condition=non exclu & non coupure & non sommet de zone def.

Cette variable condition permet de tester si le parent considéré n'est ni exclu, ni une coupure, ni le sommet d'une zone défaillante.

A l'étape 2410 la valeur booléenne de la variable condition est évaluée. Si celle-ci est fausse, c'est-à-dire que le parent considéré est soit exclu, soit une coupure soit le sommet d'une zone de définition, un nouveau parent est proposé. Si la variable condition est vraie, c'est-à-dire que le nœud considéré n'est pas exclu, n'est pas une coupure et n'est pas le sommet d'une zone défaillante, un nouvel appel récursif à la fonction **construire_liste_candidats ()** avec pour variable le parent considéré est effectué à l'étape 2412. Un nouveau parent est à nouveau proposé à l'étape 2404 à l'issue de l'étape 2412.

Lorsqu'il n'est plus possible de proposer des parents, la réponse au test de l'étape 2406 est négatif. Dans ce cas, il est vérifié à l'étape 2414 si le sommet testé est ou non mauvais. Si tel est le cas, le sommet est enregistré à l'étape 2416 avant l'achèvement de la fonction. Sinon, la fonction **construire_liste_candidats ()** s'achève sans enregistrement du sommet considéré.

L'analyse sous un sommet effectuée à l'étape ultérieure 242 de la figure 2 correspondant à l'étape 2326 de la figure 23 consiste à déterminer si les coupures se trouvant sous un sommet peuvent être résolues.

Dans un premier temps, on établit à l'étape 244 la liste des coupures, on détecte la présence éventuelle de nœuds non testables et la présence éventuelle d'un effet gigogne.

Une fois cette liste établie, on tente à l'étape 246 de résoudre les coupures. En fonction du résultat obtenu, on teste le critère d'arrêt spécifié et on retourne au contexte d'appel.

La détermination de la décision se fait de la façon suivante :

- Si le critère d'arrêt est « une cellule défaillante », décision = 1 si, à la fois, toutes les coupures sont résolues, tous les nœuds internes de la zone sont testables et s'il n'y a pas d'effet gigogne. Dans le cas contraire, décision = 0.
- Si le critère d'arrêt est « une zone défaillante », décision = 1 si, à la fois, toutes les coupures sont résolues et s'il n'y a pas d'effet gigogne. Décision = 0 dans le cas contraire.

- Si le critère d'arrêt est « tout », décision = 0 quelle que soit l'analyse.

Le détail de l'algorithme d'analyse sous un nœud sommet constituant la fonction **analyser_sous_ce_sommet( )** est donné sur la figure 25.

La fonction **analyser_sous_ce_sommet ()** reçoit en entrée le sommet 2502 de l'arbre considéré ainsi qu'un critère d'arrêt 2504.

La première étape 2506 consiste à construire la liste des coupures, analyser la testabilité de celle-ci et détecter l'éventualité d'un "effet gigogne". Cette étape est réalisée par la mise en œuvre de la **fonction lister_coupures_a_resoudre ()** qui sera décrite en détail en regard de la figure 26.

A l'issue de cette étape, les coupures sont résolues à l'étape 2508 et la liste des coupures est libérée. Cette résolution des coupures s'effectue par la fonction **resoudre_coupures ()** qui sera décrite en détail en regard de la figure 27.

A l'étape 2510 est définie une variable logique T. Celle-ci est définie par :

T=coupures toutes résolues & pas d'effet gigogne

Un test est effectué à l'étape 2512 sur la valeur de la variable logique T. Si celle-ci est vraie, c'est-à-dire que toutes les coupures peuvent être résolues et qu'il n'y a pas d'effet gigogne, le sommet considéré en 2502 est enregistré dans la liste des zones défaillantes à l'étape 2514. A l'étape 2516, on effectue l'opération suivante :

sommet->sommet_zone_def=1.

A l'issue de l'étape 2516, ou si la variable T est fausse à l'issue de l'étape 2510, une décision quant à l'analyse est prise à l'étape 2518 suivant la méthode de détermination exposée précédemment. Ensuite, cette décision est retournée de l'étape 2520 en tant que réponse à la fonction **analyser_sous_ce_sommet ()**.

Pour établir la liste des coupures à l'étape 244 de la figure 2 correspondant à l'étape 2506 de la figure 25, on parcourt l'arbre à analyser avec la procédure à ordre préfixé. Dès qu'on trouve un sommet de zone défaillante, un indicateur d'effet gigogne est mis à un. De même, dès qu'un nœud non-

testable est rencontré, un indicateur de testabilité est mis à zéro. Dès qu'on rencontre une coupure mauvaise ou non-testable, on la rajoute en tant qu'élément dans la liste et on incrémente le compteur de coupures. La programmation est récursive. Le nouveau sommet est le nœud associé au parent en cours et les paramètres nombre_de_coupures, testabilité et gigogne sont passés par référence pendant les récurrences.

Au premier appel, le nombre_de_coupures doit être initialisé à zéro, testabilité doit être initialisé à un et gigogne doit être initialisé à zéro. Au premier appel, sommet n'est ni exclu, ni coupure, ni sommet de zone défaillante, mais testable et testé mauvais.

L'algorithme détaillé de la construction de la liste des coupures constituant la fonction **lister_coupures_a_resoudre** () est décrit dans l'organigramme de la figure 26.

La fonction **lister_coupures_a_resoudre** () reçoit en entrée le sommet 2602 de l'arbre considéré, le nombre de coupures 2604, une variable de testabilité 2606 et une variable d'effet gigogne 2608.

Lors de la première étape 2610, un parent du nœud considéré est proposé. Si lors du test effectué à l'étape 2612 il est constaté que ce parent n'existe pas, il est mis un terme à la fonction à l'étape 2614. Si ce parent existe, un test est effectué à l'étape 2616 pour déterminer si le parent de ce nœud est ou non un sommet de zone défaillante. Si tel est le cas, la variable gigogne est mise à 1 à l'étape 2618. Sinon, la variable gigogne est maintenue à sa valeur initiale.

A l'étape ultérieure 2620, il est vérifié s'il s'agit d'un parent exclu ou du sommet d'une zone défaillante. Si la réponse est positive, l'étape 2610 est mise en œuvre pour proposer un autre parent du nœud considéré. Si la réponse est négative, il est vérifié à l'étape 2622 s'il s'agit d'une coupure mauvaise ou non testable. Si tel est le cas, le nombre de coupures est incrémenté à l'étape 2624 puis cette coupure est ajoutée à la liste des coupures à l'étape 2626. L'étape 2610 est enfin mise en œuvre pour proposer un nouveau parent du nœud considéré.

En revanche, si à l'étape 2622, il est constaté qu'il ne s'agit ni d'une coupure mauvaise ni d'un nœud non testable, la variable testabilité est mise

à 0 à l'étape 2628. Un nouvel appel à la fonction **lister_coupures_a_resoudre ()** est effectué par récursivité à l'étape 2630 en considérant comme sommet le parent déterminé précédemment à l'étape 2610.

A l'issue de cet appel récursif de la fonction, l'étape 2610 est à nouveau mise en œuvre pour proposer un nouveau parent au nœud considéré.

Le processus décrit ici s'effectue jusqu'à achèvement des parents du nœud considéré.

Pour la résolution des coupures à l'étape 246 de la figure 2, correspondant à l'étape 2508 de la figure 25, on prend chaque coupure de la liste des coupures à résoudre et on cherche si on peut la résoudre dans le sous-arbre dans lequel on fait l'analyse.

Pour décider si la résolution est possible, on applique le critère de résolution énoncé précédemment.

Dans un premier temps, on essaie de résoudre la coupure sur le sommet. Si cette tentative échoue, on recherche une résolution dans le reste de l'arbre avec une procédure récursive. Cette procédure récursive est décrite dans la suite.

La procédure générale de résolution des coupures est détaillée dans l'organigramme de la figure 27 décrivant l'algorithme de la fonction **resoudre_coupures( )**.

La fonction **resoudre_coupures ()** reçoit en entrée la liste des coupures 2702, le nombre des coupures 2704, ainsi que le sommet de l'arbre considéré 2706. Initialement, à l'étape 2708 la variable compteur résolution est mise à 0.

A l'étape 2710, une coupure est proposée dans la liste. Il est vérifié à l'étape 2712 que cette coupure existe. Si tel est le cas, il est déterminé à l'étape 2714 si cette coupure est mauvaise et si celle-ci est raccordée à un sommet. Si tel est le cas, le compteur de résolution est incrémenté à l'étape 2716 et l'étape 2710 est à nouveau mise en œuvre pour proposer une nouvelle coupure.

En revanche, si la coupure proposée n'est pas une coupure mauvaise qui est raccordée à un sommet, l'étape 2718 est mise en œuvre. Celle-ci

consiste à rechercher un nœud solution sous le sommet considéré. Cette recherche s'effectue par appel de la fonction **chercher_un_nœud ()** qui sera décrite en regard de la figure 28.

Il est vérifié à l'étape suivante 2720 si un tel nœud a pu être trouvé. Si tel est le cas, le compteur de résolution est incrémenté à l'étape 2722. A l'issue de l'étape 2722, ou dans le cas où aucun nœud n'a été trouvé lors de la recherche 2718, une nouvelle coupure est proposée dans la liste à l'étape 2710.

Si le test opéré à l'étape 2712 révèle qu'il n'existe plus de coupures dans la liste, le compteur de résolution est comparé au nombre de coupures à l'étape 2724. Si ceux-ci sont égaux, la variable R est fixée à 1 à l'étape 2726. Sinon, celle-ci est fixée à la valeur 0 à l'étape 2728. La variable R indique si oui ou non la résolution a été effectuée. Enfin, la variable R est retournée comme solution de la fonction **resoudre_coupure ()** à l'étape 2730.

La recherche d'un noeud solution effectuée à l'étape 2718 et constituant la fonction **chercher_un_nœud( )** est décrite sur la figure 28.

La fonction **chercher_un_nœud ()** reçoit en entrée la coupure considérée 2802 ainsi que le sommet de l'arbre considéré 2804.

Initialement, la valeur de la variable test R est fixée à 0 à l'étape 2806. Un parent du sommet considéré est proposé à l'étape 2810. Il est vérifié à l'étape 2812 si ce parent existe. Si tel n'est pas le cas, la valeur de la variable R est fixée à 0 à l'étape 2814 et la variable R est retournée à l'étape 2816 en tant que résultat de la fonction **chercher_un_nœud ()**.

Si le parent existe à l'étape 2812, il est déterminé à l'étape 2818 si ce parent est une coupure. Si tel est le cas, un nouveau parent est proposé à l'étape 2810.

Si tel n'est pas le cas, plusieurs tests sont effectués successivement. Lorsque l'un de ces tests est vérifié, la variable R est fixée à la valeur 1 à l'étape 2820. Le premier test opéré à l'étape 2822 consiste à vérifier si le nœud considéré est une coupure connectée à un nœud exclu. Si la réponse est négative, il est ensuite déterminé à l'étape 2824 si ce nœud est une coupure connectée à un nœud non testable et n'est pas une coupure. Si la réponse est négative, il est déterminé à l'étape 2826 si ce nœud est une cou-

pure connectée à un nœud mauvais et ne constituant pas le sommet d'une zone défaillante. Si la réponse est toujours négative, il est déterminé à l'étape 2828 si, pour le nœud considéré, les parents ne sont pas une coupure, s'il est non exclu et s'il n'est pas le sommet d'une zone défaillante. Si tel est le cas, un nouvel appel récursif de la fonction **chercher_un_nœud ()** est effectué à l'étape 2830. Lors de cet appel récursif, le nouveau sommet considéré est le parent pointant vers le nœud.

Si à l'issue de l'étape 2828 ou 2830 la valeur de la variable test R est égale à 1, lors d'une étape de vérification notée 2832, l'étape 2820 est à nouveau mise en œuvre. Si la valeur de la variable test n'est pas égale à 1, un nouveau parent du sommet est proposé à l'étape 2810.

REVENDICATIONS

1.- Procédé de localisation d'un élément défectueux dans un circuit intégré dont l'agencement théorique est connu, du type comportant une succession d'étapes consistant en :

- la détermination d'un point de mesure du circuit intégré ; et
- le test du point de mesure déterminé par mise en œuvre de :
  - l'application d'une séquence de tests aux entrées du circuit intégré ;
  - la mesure de signaux au point de mesure déterminé du circuit intégré, lors de l'application de la séquence de tests ; et
  - l'évaluation du point de mesure par la comparaison des signaux mesurés avec des signaux théoriques devant être obtenus au point de mesure déterminé pour évaluer si le point de mesure est défaillant ou satisfaisant ; et

dans lequel la position de l'élément défectueux du circuit intégré est déterminée à partir des évaluations effectuées aux différents points de mesure déterminés,

caractérisé en ce qu'il comporte initialement :

- une étape de modélisation de l'agencement théorique du circuit intégré, sous forme d'au moins un graphe comportant un ensemble de nœuds et d'arcs orientés des entrées du circuit vers les sorties du circuit ;
- considérer comme un sous-graphe de recherche , un sous-graphe dont le nœud formant sommet correspond à un point de mesure défaillant ;

- et en ce que, pour la recherche de l'élément défectueux, il comporte les étapes de :

- affecter à chaque nœud du sous-graphe de recherche considéré, une variable caractéristique dépendant de la structure du sous-graphe de recherche ;

- considérer comme point de mesure, le point de mesure correspondant à un nœud du sous-graphe considéré, obtenu par application d'un cri-

tère prédéterminé portant sur les variables caractéristiques de l'ensemble des nœuds du sous-graphe de recherche considéré ;

- effectuer un test du point de mesure considéré ;
- considérer comme nouveau sous-graphe de recherche :

soit, le sous-graphe de recherche préalablement considéré en excluant le nœud correspondant au point de mesure testé et tous ses nœuds parents, si le point de mesure est satisfaisant

soit, un sous-graphe dont le nœud correspondant au point de mesure est le sommet , si le point de mesure est défaillant ; et

- rechercher, dans le nouveau sous-graphe de recherche considéré l'élément défectueux, jusqu'à vérification d'un critère d'arrêt prédéterminé.

2.- Procédé de localisation selon la revendication 1, caractérisé en ce que, lors de l'étape initiale de modélisation de l'agencement théorique du circuit intégré, le circuit est modélisé sous forme d'un arbre par création éventuelle de nœuds virtuels (810 ; 2208, 2210, 2218) lorsque un même nœud est le parent de au moins deux nœuds, eux-mêmes parents d'un même nœud.

3.- Procédé de localisation selon la revendication 2, caractérisé en ce qu'il comporte, après vérification du critère d'arrêt prédéterminé, les étapes de :

- évaluer dans le ou chaque dernier sous-graphe de recherche, si pour chaque nœud virtuel correspondant à un point de mesure défaillant, le nœud jumeau associé audit nœud virtuel est un nœud du même sous graphe correspondant également à un point de mesure défaillant ; et
- considérer alors le ou chaque sous-groupe pour lequel la condition est vérifiée comme correspondant à une partie du circuit intégré comportant au moins un élément défectueux.

4.- Procédé de localisation selon l'une quelconque des revendications précédentes, caractérisé en ce que ladite variable caractéristique propre à chaque nœud est le nombre d'ancêtres de ce nœud dans le sous-graphe de recherche considéré.

5.- Procédé de localisation selon la revendication 4 prise avec l'une des revendications 2 et 3, caractérisé en ce que ledit critère prédéterminé

est adapté pour déterminer le nœud dont le nombre d'ancêtres est sensiblement égal au nombre moyen d'ancêtres par nœud dans le sous-arbre de recherche considéré.

6.- Procédé de localisation selon l'une quelconque des revendications précédentes, caractérisé en ce qu'il comporte une étape d'affectation à chaque nœud, d'un indicateur de conformité initialement fixé à un état défaillant ; et

en ce que, pour la détermination du nouveau sous-graphe de recherche à considérer, il comporte les étapes de :

- fixer l'indicateur de conformité du nœud correspondant au point de mesure testé et de tous ses nœuds parents à un état satisfaisant, si le point de mesure testé est satisfaisant ; et

- considérer comme nouveau sous-graphe de recherche, le sous-graphe inclus dans le sous-graphe de recherche précédent et comportant les seuls nœuds dont l'indicateur de conformité est fixé à l'état défaillant.

7.- Procédé de localisation selon l'une quelconque des revendications précédentes, caractérisé en ce que le sous-graphe de recherche initialement considéré est formé de l'intersection des sous-graphes ayant chacun pour sommet un nœud correspondant à une sortie défaillante du circuit intégré.

8.- Dispositif de localisation d'un élément défectueux dans un circuit intégré dont l'agencement théorique est connu, du type comportant des moyens pour effectuer une succession d'étapes consistant en :

- la détermination d'un point de mesure du circuit intégré ; et

- le test du point de mesure déterminé par mise en œuvre de :

- l'application d'une séquence de tests aux entrées du circuit intégré ;

- la mesure de signaux au point de mesure déterminé du circuit intégré, lors de l'application de la séquence de tests ; et

- l'évaluation du point de mesure par la comparaison des signaux mesurés avec des signaux théoriques devant être obtenus au point de mesure déterminé pour évaluer si le point de mesure est défaillant ou satisfaisant ; et

des moyens pour déterminer la position de l'élément défectueux du circuit intégré à partir des évaluations effectuées aux différents points de mesure déterminés,

caractérisé en ce qu'il comporte :

- des moyens de modélisation initiale de l'agencement théorique du circuit intégré, sous forme d'au moins un graphe comportant un ensemble de nœuds et d'arcs orientés des entrées du circuit vers les sorties du circuit ;
- des moyens pour considérer initialement comme un sous-graphe de recherche , un sous-graphe dont le nœud formant sommet correspond à un point de mesure défaillant ;

- et en ce que, pour la recherche de l'élément défectueux, il comporte des moyens pour :

- affecter à chaque nœud du sous-graphe de recherche considéré, une variable caractéristique dépendant de la structure du sous-graphe de recherche ;

- considérer comme point de mesure, le point de mesure correspondant à un nœud du sous-graphe considéré, obtenu par application d'un critère prédéterminé portant sur les variables caractéristiques de l'ensemble des nœuds du sous-graphe de recherche considéré ;

- effectuer un test du point de mesure considéré ;

- considérer comme nouveau sous-graphe de recherche :

soit, le sous-graphe de recherche préalablement considéré en excluant le nœud correspondant au point de mesure testé et tous ses nœuds parents, si le point de mesure est satisfaisant

soit, un sous-graphe dont le nœud correspondant au point de mesure est le sommet , si le point de mesure est défaillant ; et

- rechercher, dans le nouveau sous-graphe de recherche considéré l'élément défectueux, jusqu'à vérification d'un critère d'arrêt prédéterminé.

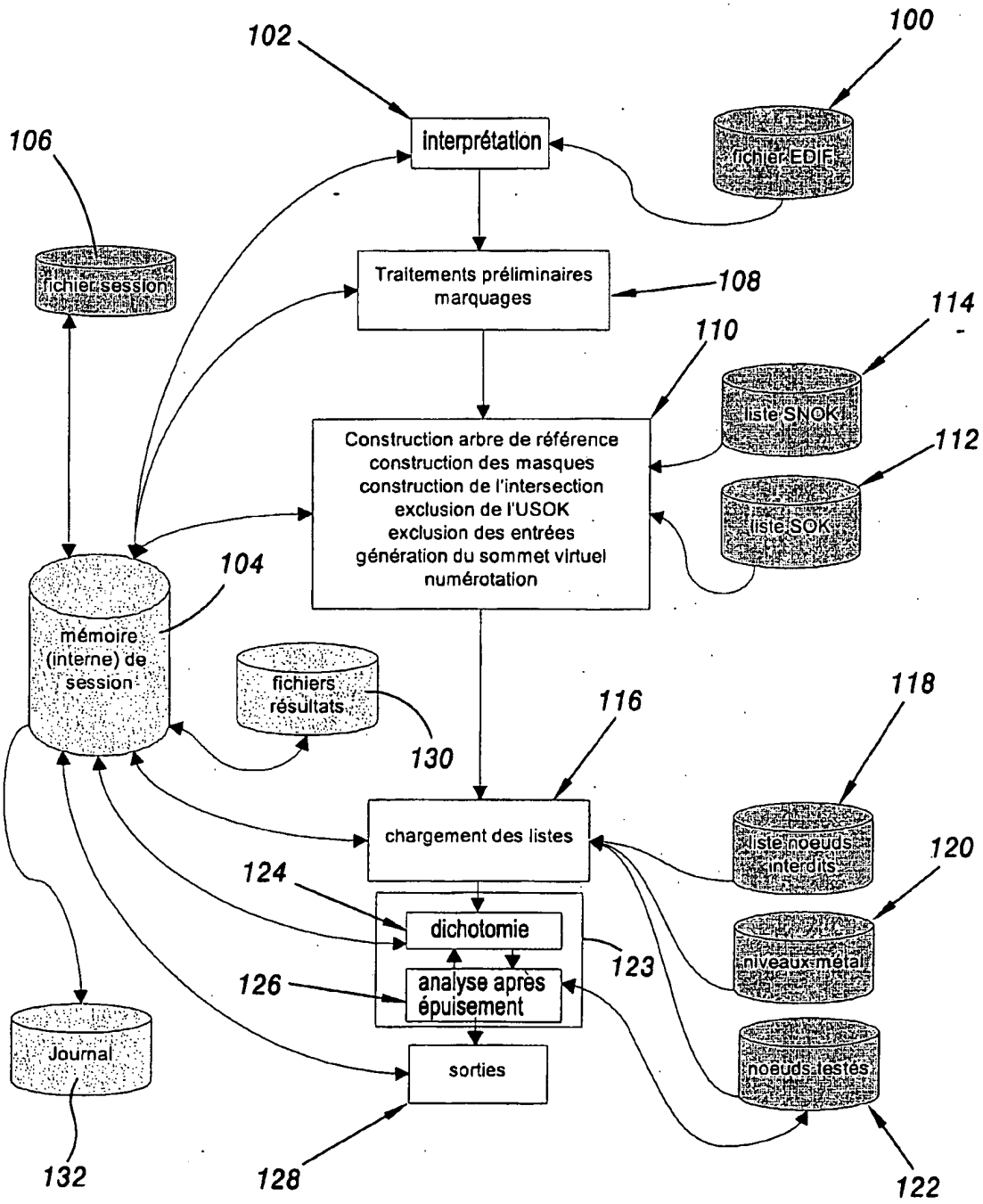

102
100
interprétation
fichier EDIF
106
fichier session
Traitements préliminaires
marquages
108
114
110
liste SNOK
Construction arbre de référence
construction des masques
construction de l'intersection
exclusion de l'USOK
exclusion des entrées
génération du sommet virtuel
numérotation
112
liste SOK
104
mémoire
(interne) de
session
fichiers
résultats
130
116
118
chargement des listes
liste noeuds
interdits
120
124
dichotomie
123
niveaux métal
126
analyse apres
épuisement
Journal
sorties
noeuds testés
132
128
122

FIG.1

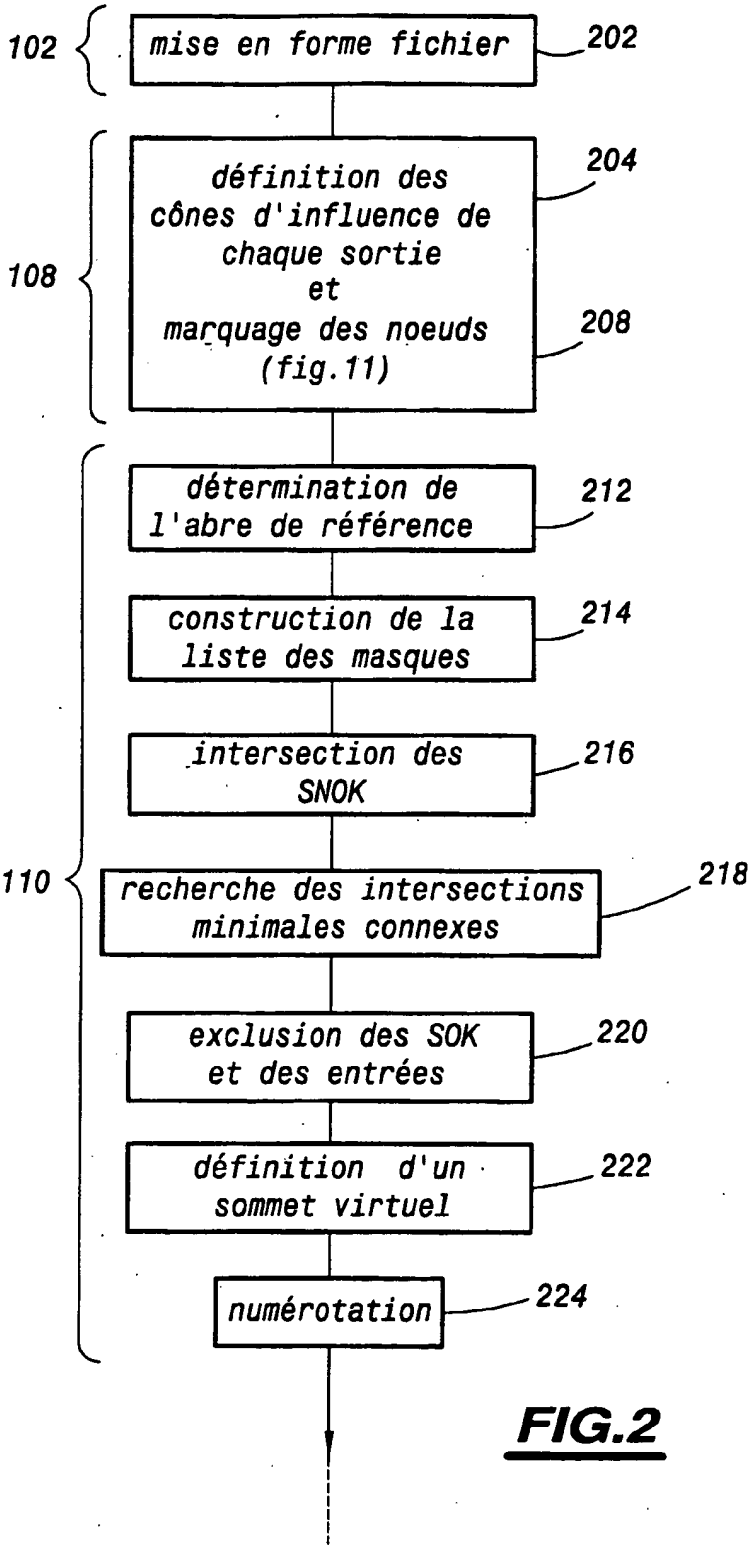

102
mise en forme fichier
202
108
définition des cônes d'influence de chaque sortie et marquage des noeuds (fig.11)
204
208
détermination de l'abre de référence
212
construction de la liste des masques
214
intersection des SNOK
216
110
recherche des intersections minimales connexes
218
exclusion des SOK et des entrées
220
définition d'un sommet virtuel
222
numérotation
224

FIG.2

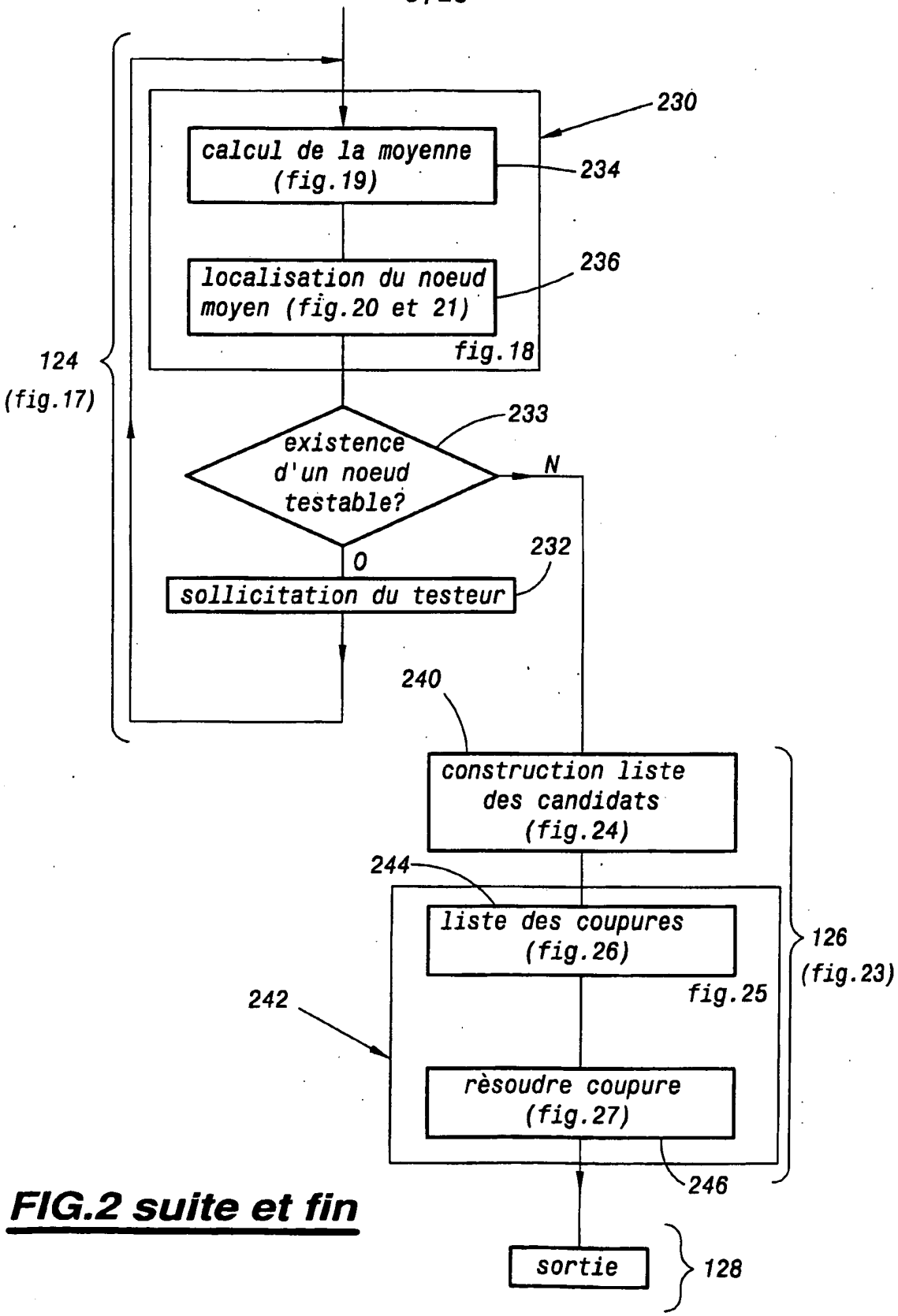


FIG.2 suite et fin

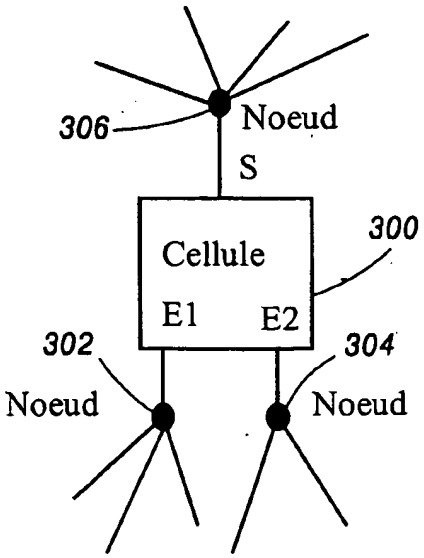


FIG.3

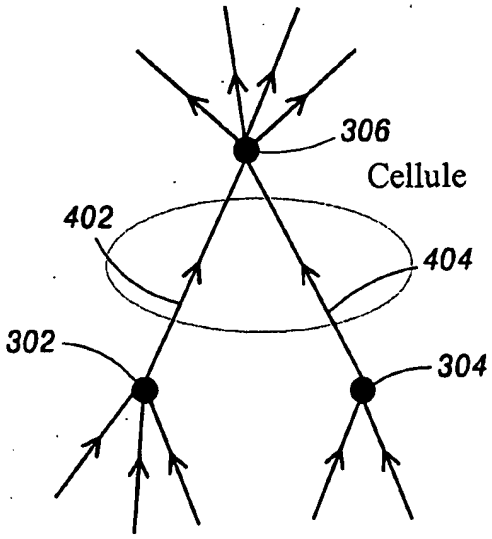


FIG.4

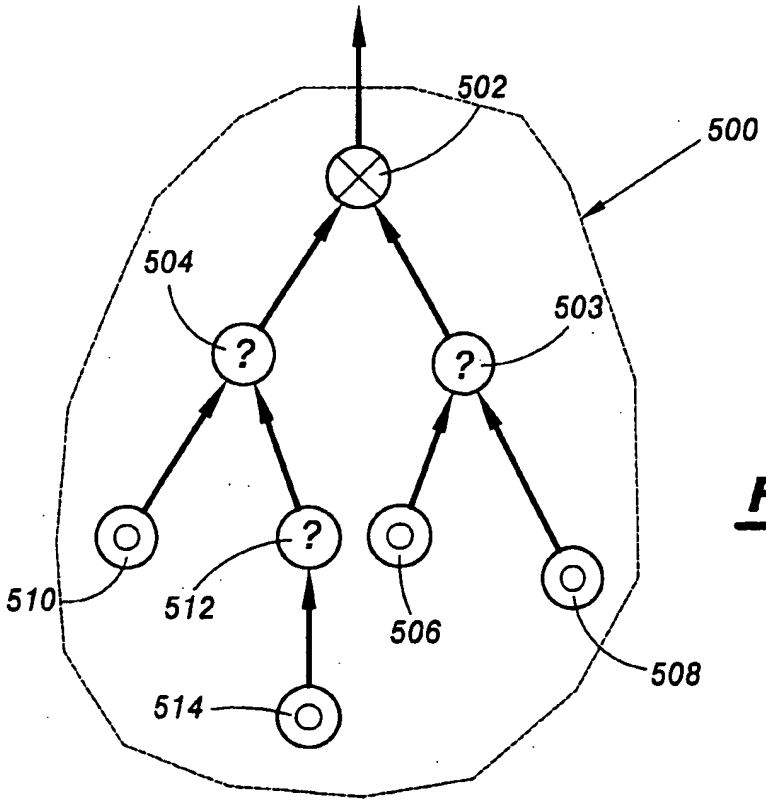


FIG.5

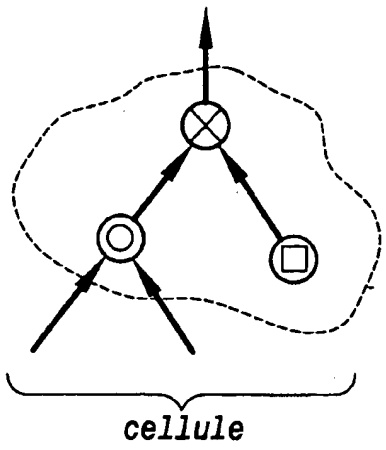


FIG.6

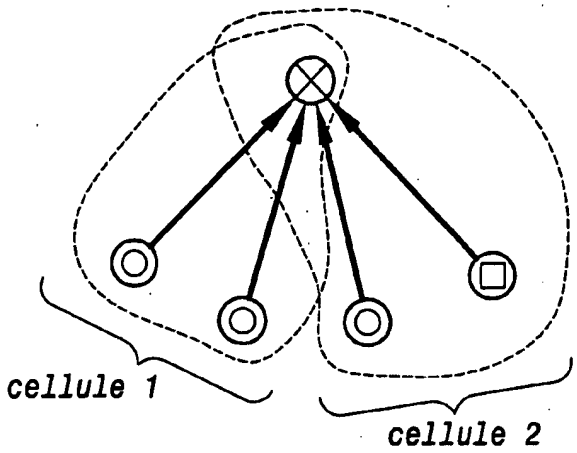


FIG.7

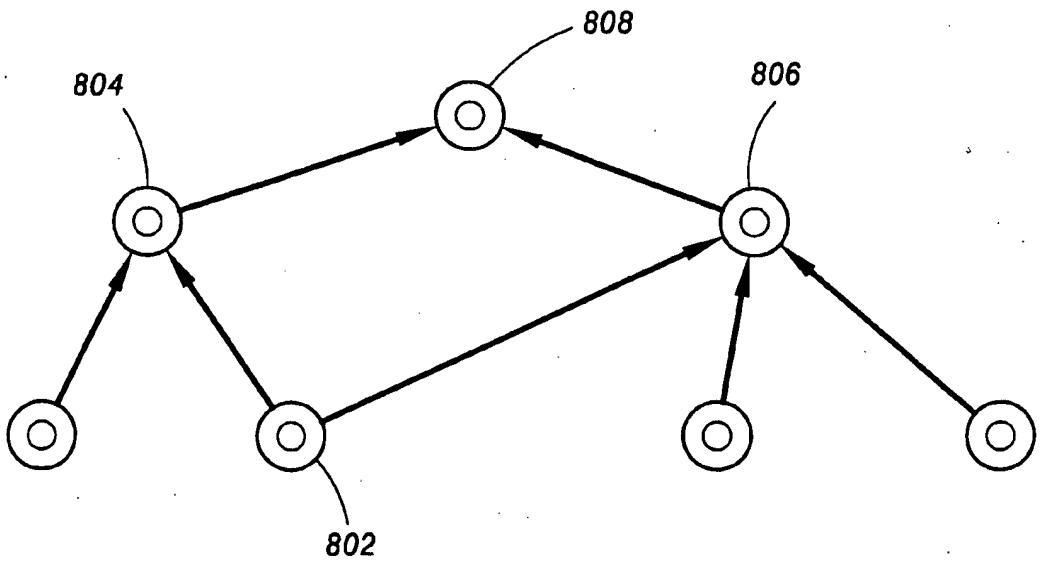


FIG.8A

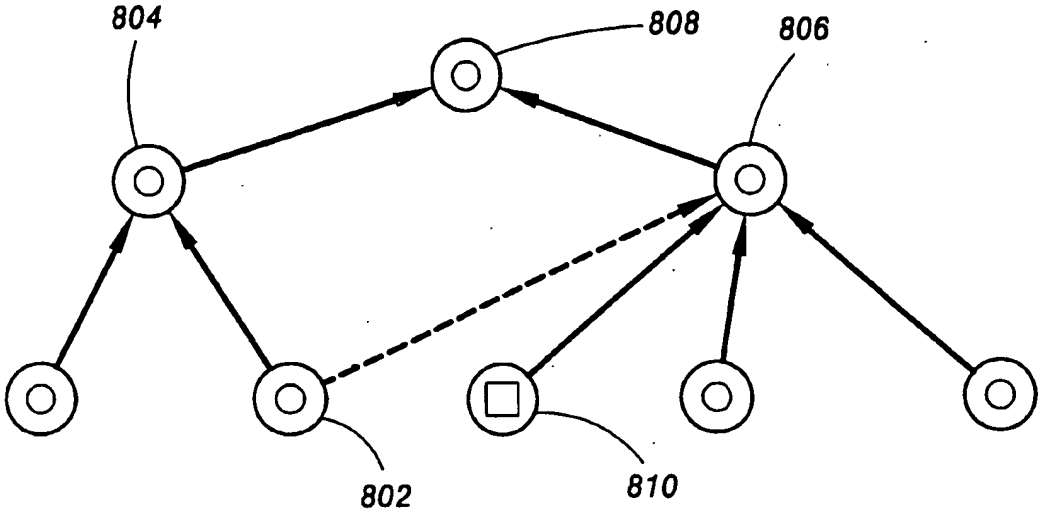

804
808
806
802
810

FIG.8B

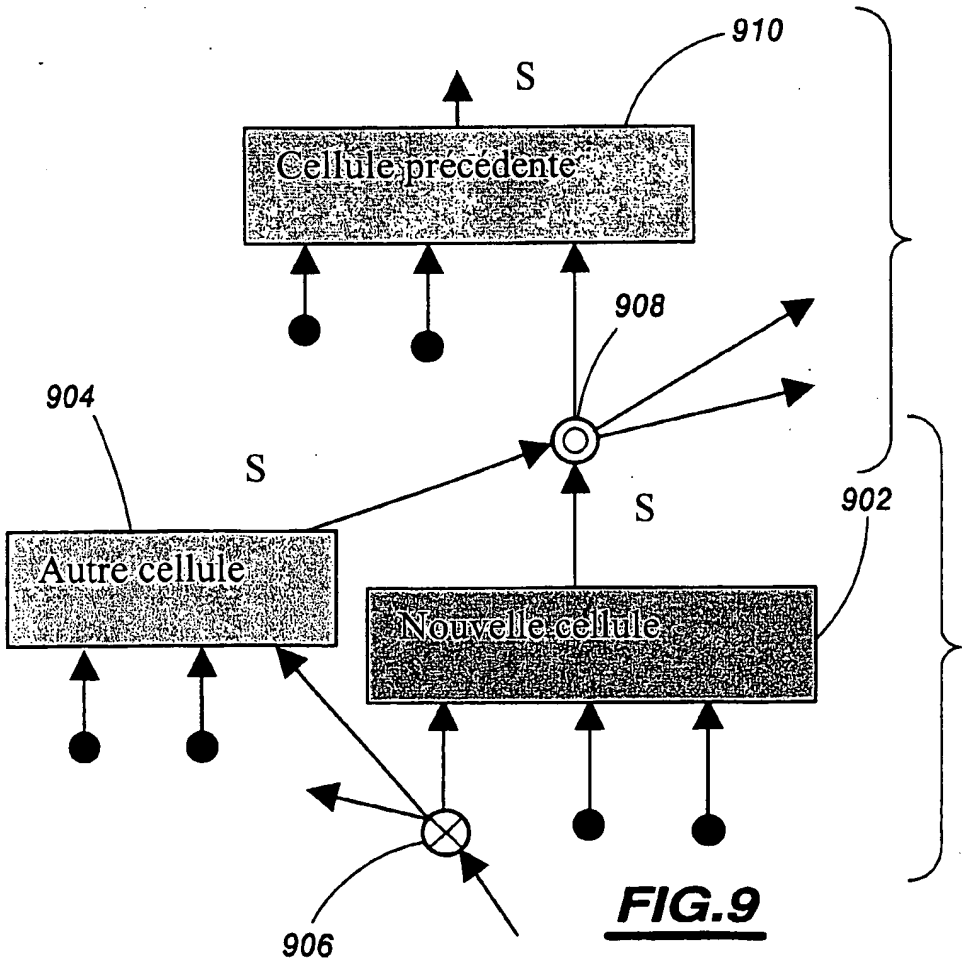

910
S
Cellule precedente
908
904
S
S
902
Autre cellule
Nouvelle cellule
906

FIG.9

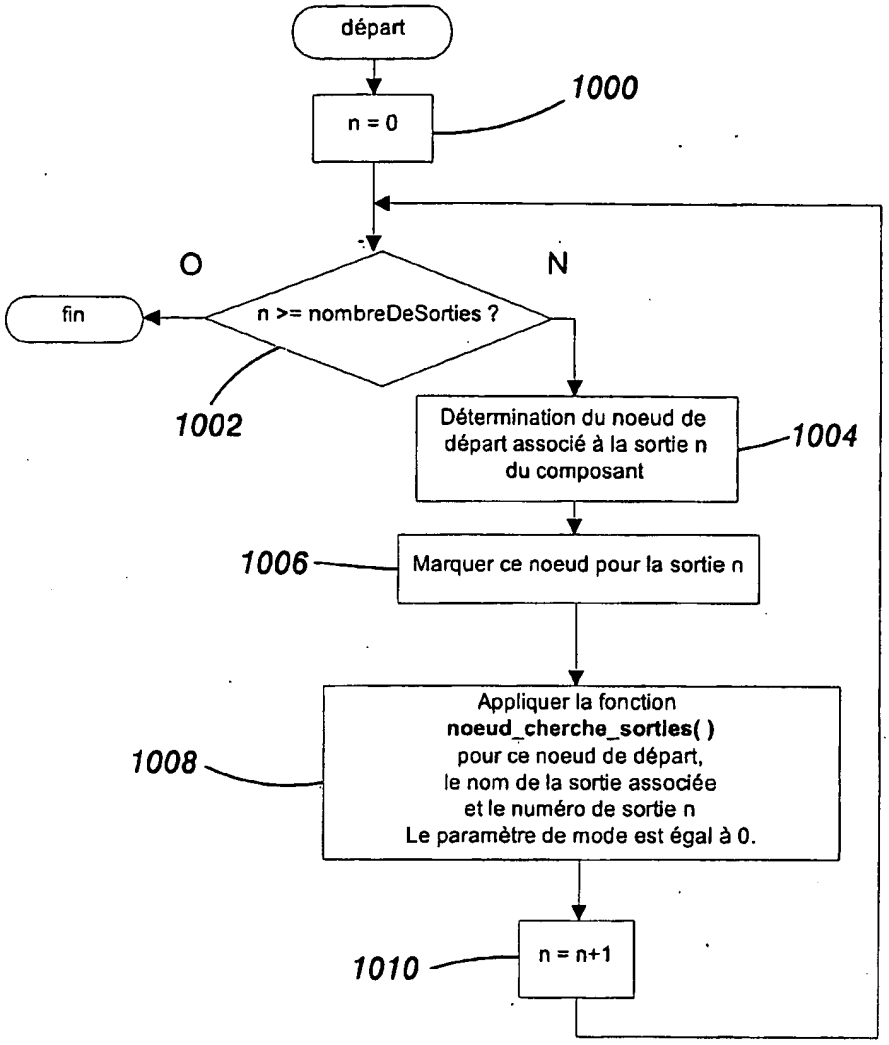

départ
1000
n = 0
O
N
fin
n >= nombreDeSorties ?
1002
Détermination du noeud de départ associé à la sortie n du composant
1004
1006
Marquer ce noeud pour la sortie n
1008
Appliquer la fonction
noeud_cherche_sorties( )
pour ce noeud de départ,
le nom de la sortie associée
et le numéro de sortie n
Le paramètre de mode est égal à 0.
1010
n = n+1

FIG.10

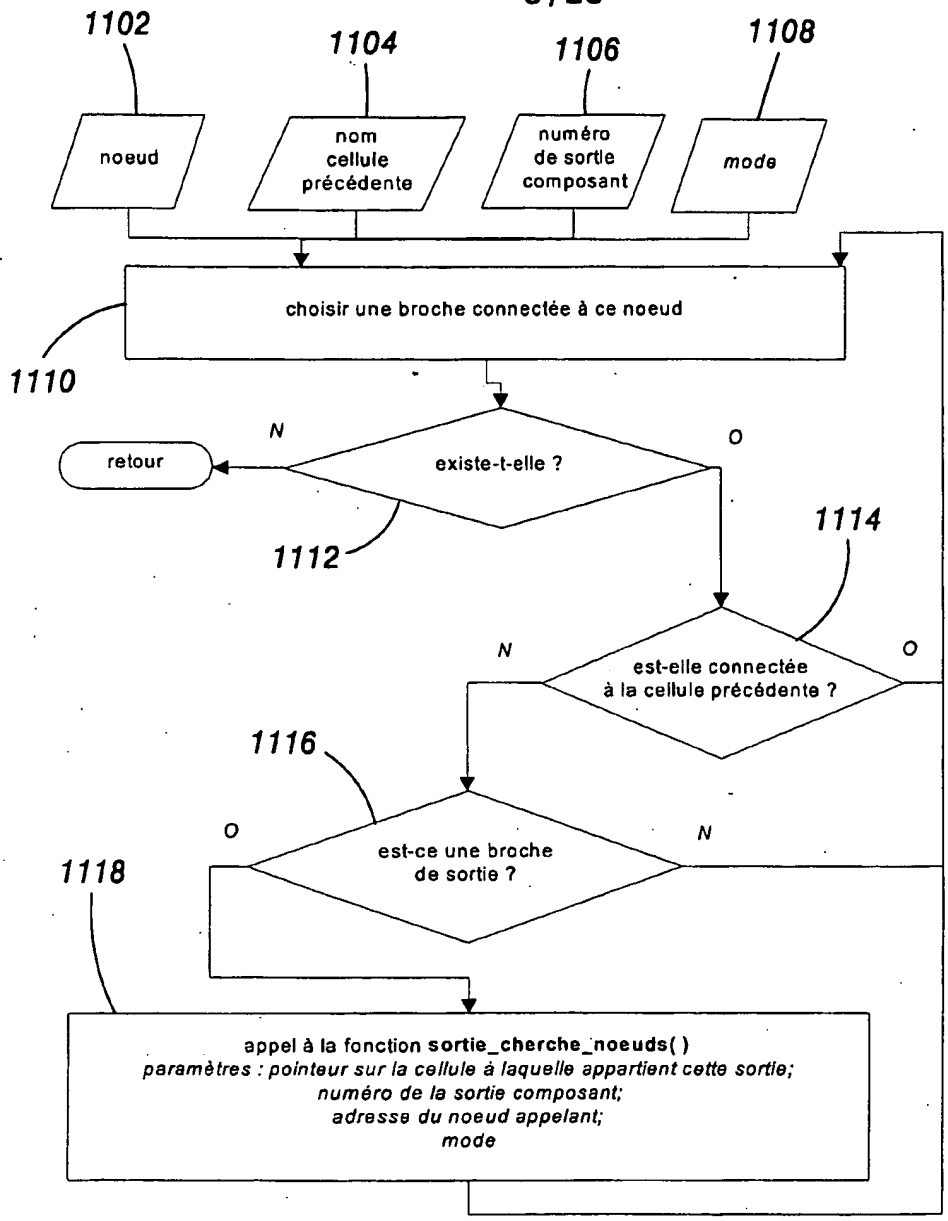

1102
1104
1106
1108
noeud
nom cellule précédente
numéro de sortie composant
mode
choisir une broche connectée à ce noeud
1110
N
O
retour
existe-t-elle ?
1112
1114
N
O
est-elle connectée à la cellule précédente ?
1116
O
N
est-ce une broche de sortie ?
1118
appel à la fonction sortie_cherche_noeuds( )
paramètres : pointeur sur la cellule à laquelle appartient cette sortie;
numéro de la sortie composant;
adresse du noeud appelant;
mode

FIG.11

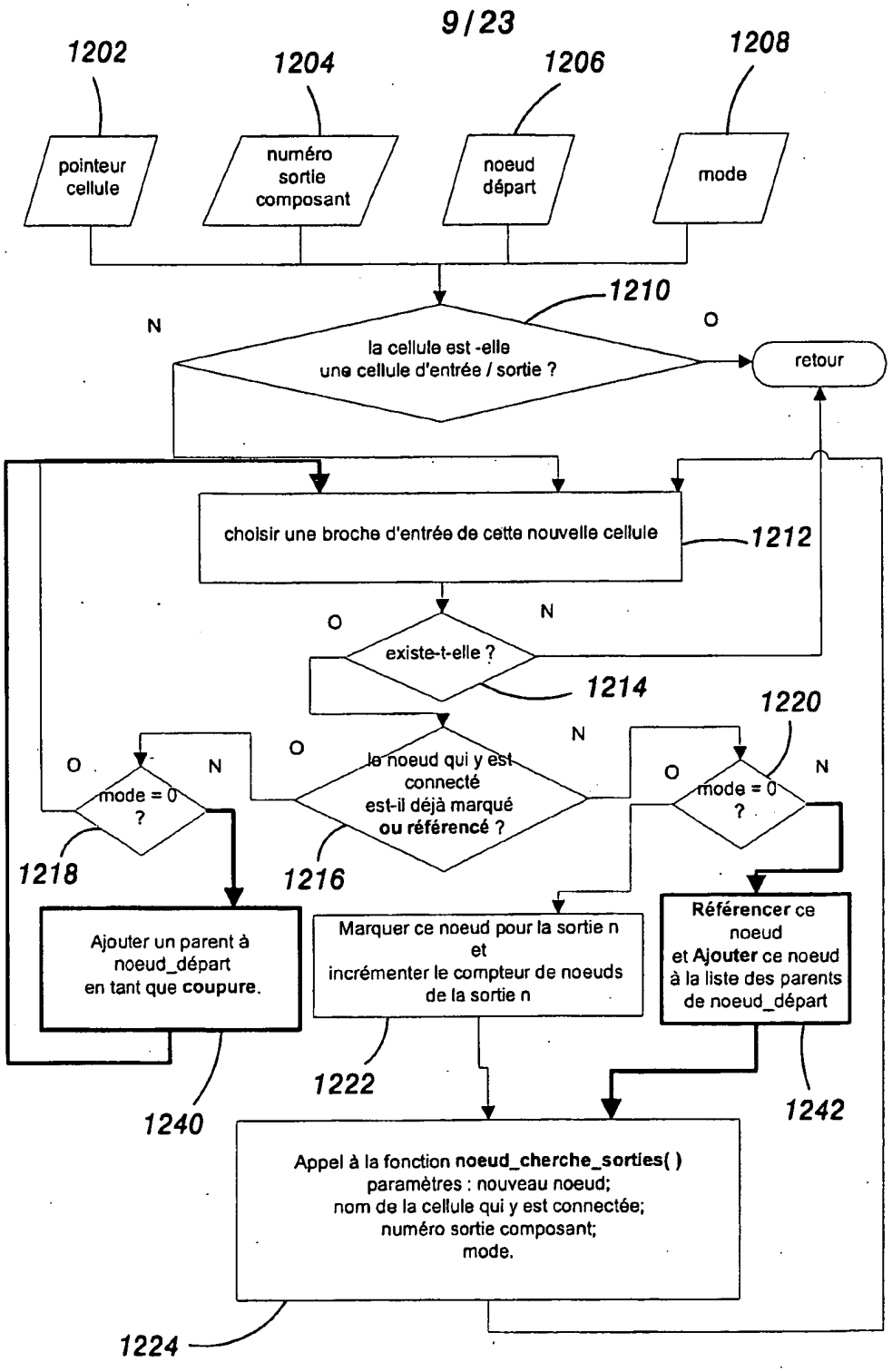

1202
1204
1206
1208
pointeur cellule
numéro sortie composant
noeud départ
mode
1210
N
O
la cellule est -elle une cellule d'entrée / sortie ?
retour
choisir une broche d'entrée de cette nouvelle cellule
1212
O
N
existe-t-elle ?
1214
1220
O
N
O
N
O
N
mode = 0 ?
le noeud qui y est connecté est-il déjà marqué ou référencé ?
mode = 0 ?
1218
1216
Ajouter un parent à noeud_départ en tant que coupure.
Marquer ce noeud pour la sortie n et incrémenter le compteur de noeuds de la sortie n
Référencer ce noeud et Ajouter ce noeud à la liste des parents de noeud_départ
1240
1222
1242
Appel à la fonction noeud_cherche_sorties( )
paramètres : nouveau noeud;
nom de la cellule qui y est connectée;
numéro sortie composant;
mode.
1224

FIG.12

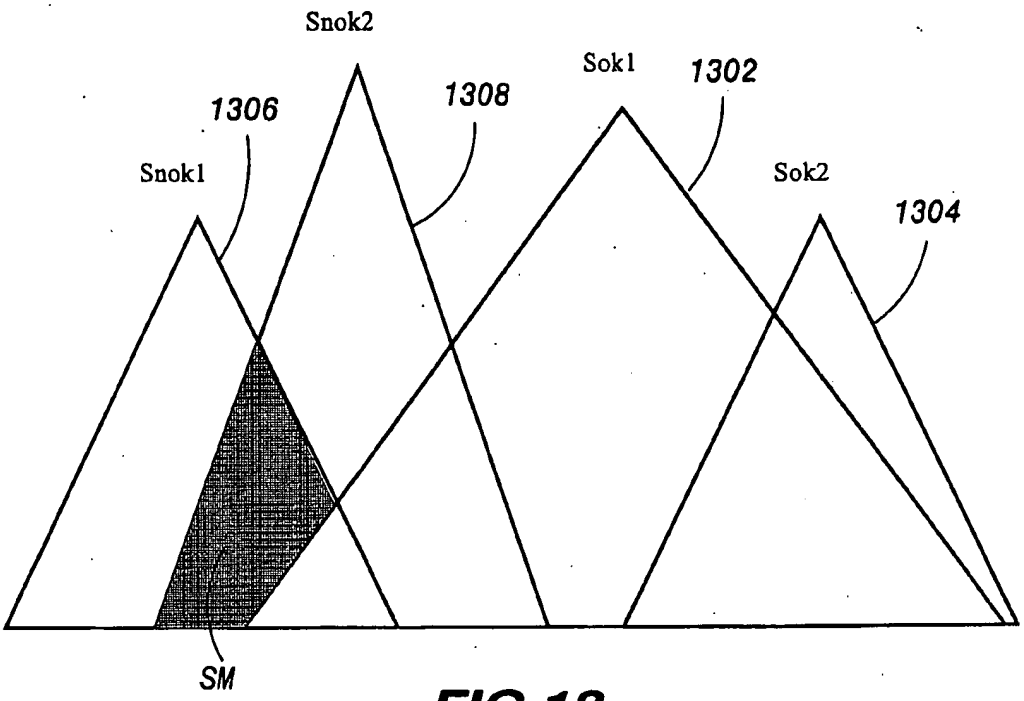

Snok2
Sok1
1302
1306
1308
Snok1
Sok2
1304
SM

FIG.13

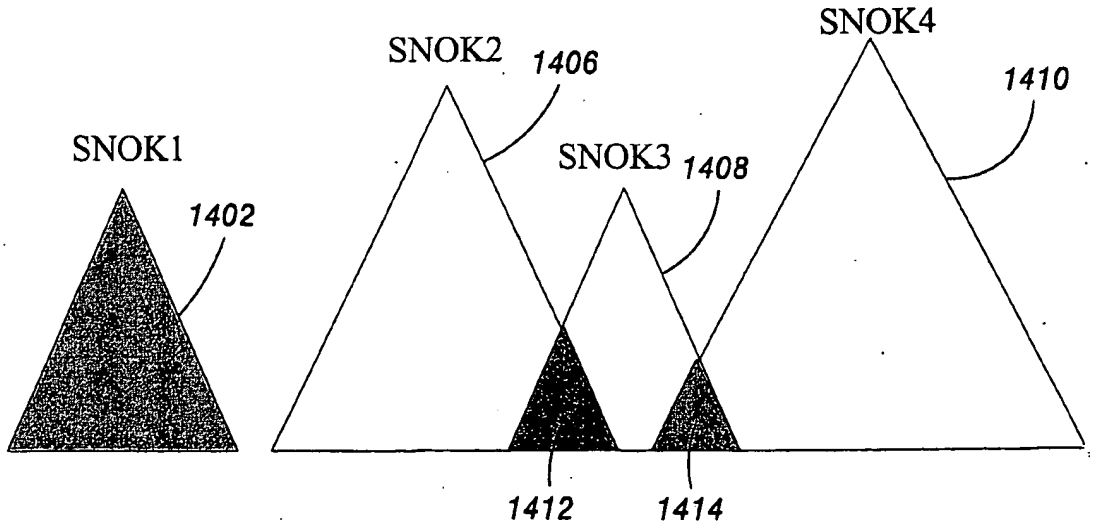

SNOK4
SNOK2
1406
1410
SNOK1
SNOK3
1408
1402
1412
1414

FIG.14

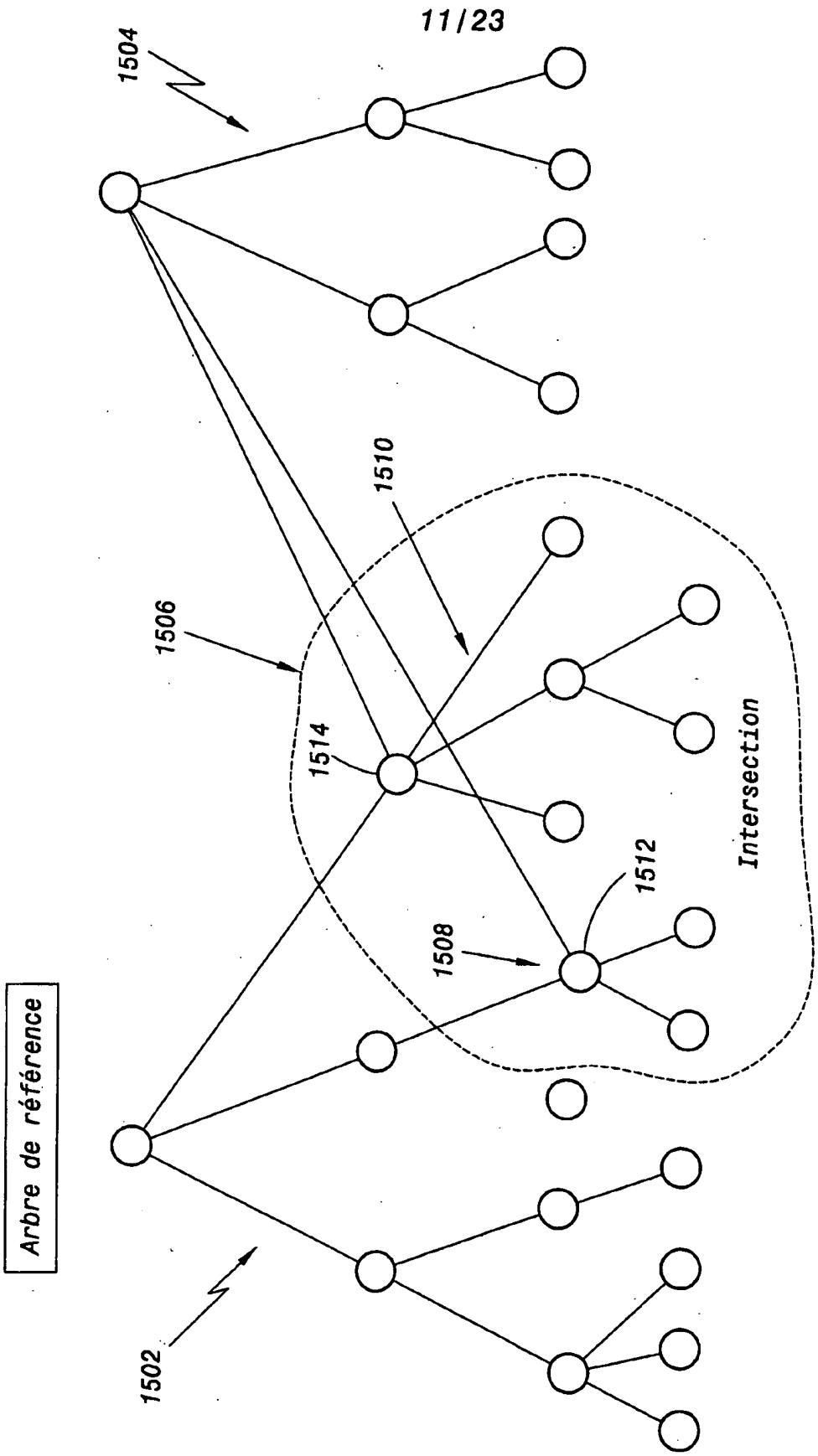
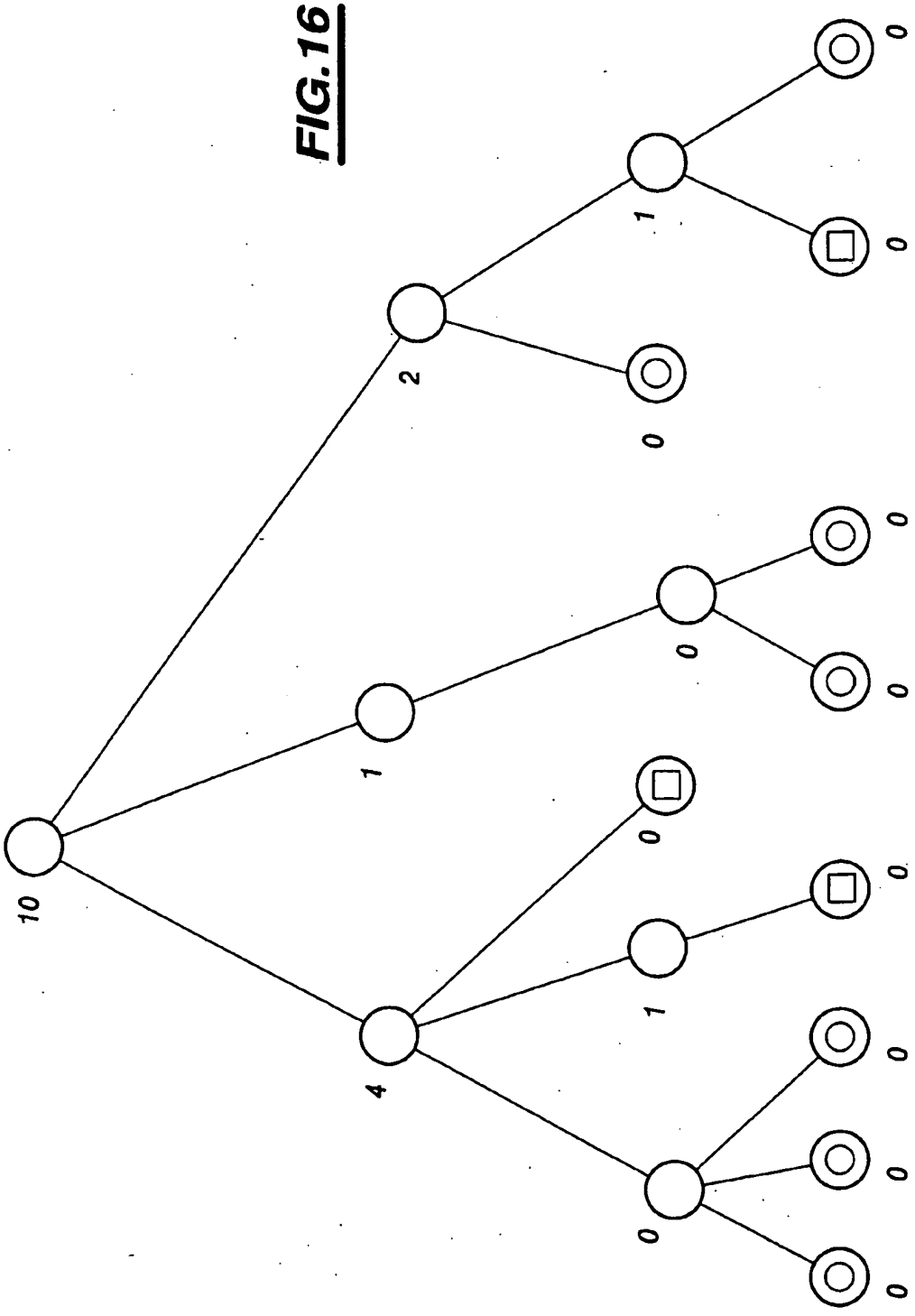

1504
1510
1506
1514
1512
1508
Intersection
Arbre de référence
1502

FIG.15

FIG.16



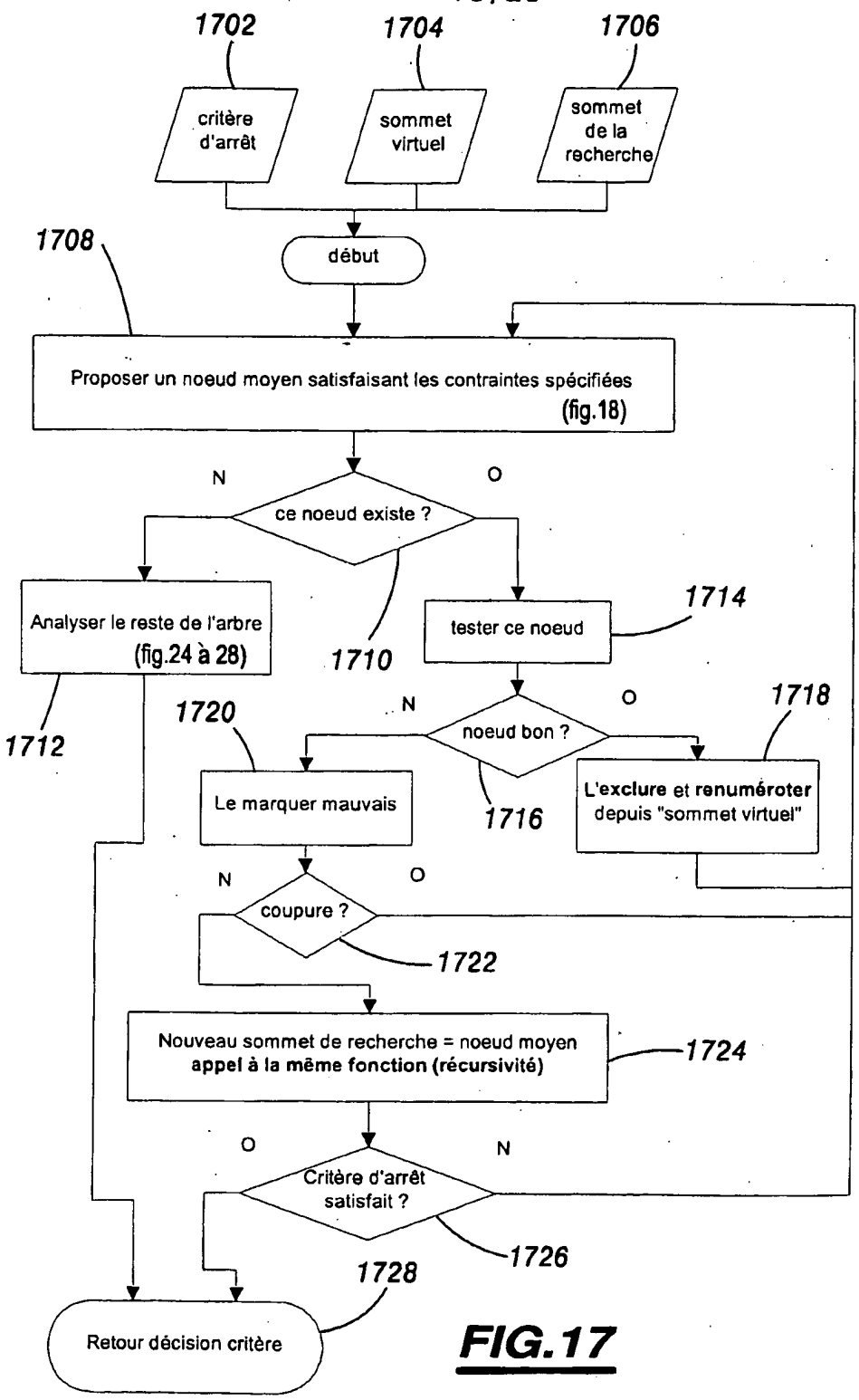

1702
1704
1706
critère d'arrêt
sommet virtuel
sommet de la recherche
début
1708
Proposer un noeud moyen satisfaisant les contraintes spécifiées (fig.18)
N
O
ce noeud existe ?
1710
Analyser le reste de l'arbre (fig.24 à 28)
1712
tester ce noeud
1714
N
O
noeud bon ?
1716
1718
1720
Le marquer mauvais
L'exclure et renuméroter depuis "sommet virtuel"
N
O
coupure ?
1722
Nouveau sommet de recherche = noeud moyen
appel à la même fonction (récursivité)
1724
O
N
Critère d'arrêt satisfait ?
1726
1728
Retour décision critère

FIG.17

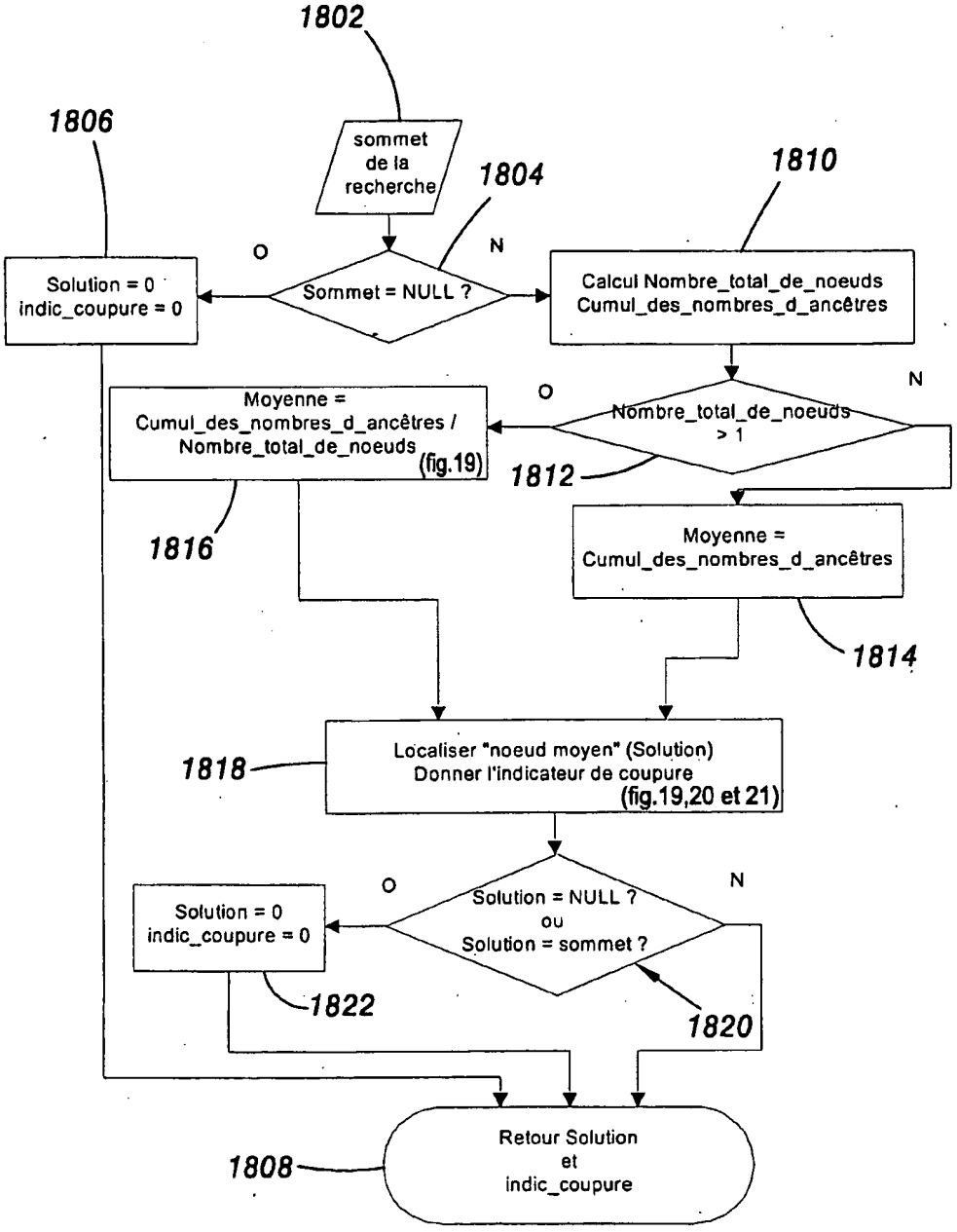

1802
sommet de la recherche
1804
1806
1810
O
N
Solution = 0
indic_coupure = 0
Sommet = NULL ?
Calcul Nombre_total_de_noeuds
Cumul_des_nombres_d_ancêtres
O
N
Moyenne = Cumul_des_nombres_d_ancêtres / Nombre_total_de_noeuds (fig.19)
Nombre_total_de_noeuds > 1
1812
1816
Moyenne = Cumul_des_nombres_d_ancêtres
1814
1818
Localiser "noeud moyen" (Solution)
Donner l'indicateur de coupure
(fig.19,20 et 21)
O
N
Solution = 0
indic_coupure = 0
Solution = NULL ?
ou
Solution = sommet ?
1822
1820
1808
Retour Solution
et
indic_coupure

FIG.18

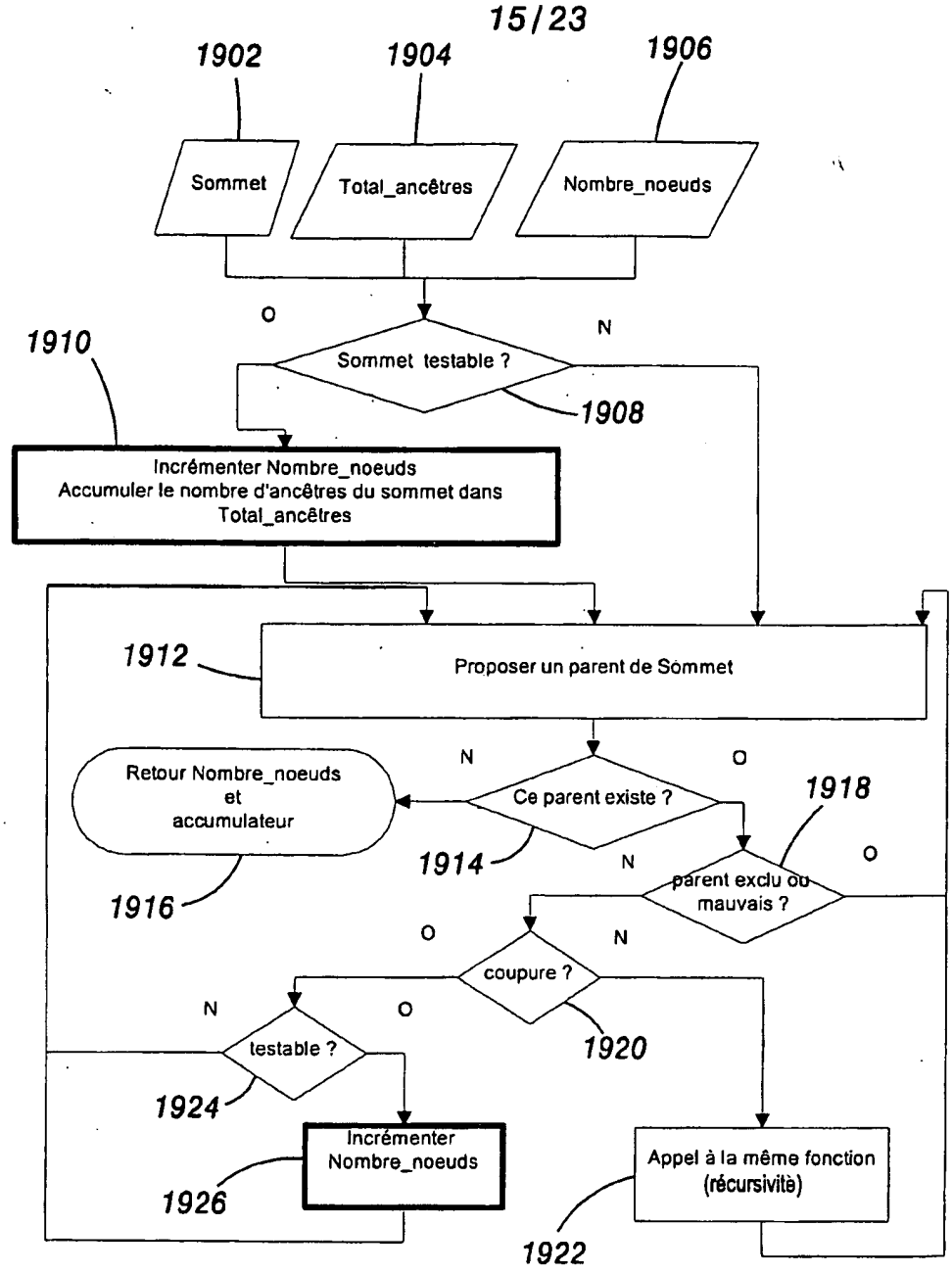

1902
1904
1906
Sommet
Total_ancêtres
Nombre_noeuds
O
N
1910
Sommet testable ?
1908
Incrémenter Nombre_noeuds
Accumuler le nombre d'ancêtres du sommet dans Total_ancêtres
1912
Proposer un parent de Sommet
N
O
Retour Nombre_noeuds et accumulateur
Ce parent existe ?
1918
1914
N
O
parent exclu ou mauvais ?
1916
O
N
coupure ?
N
O
testable ?
1920
1924
Incrémenter Nombre_noeuds
Appel à la même fonction (récursivité)
1926
1922

FIG.19

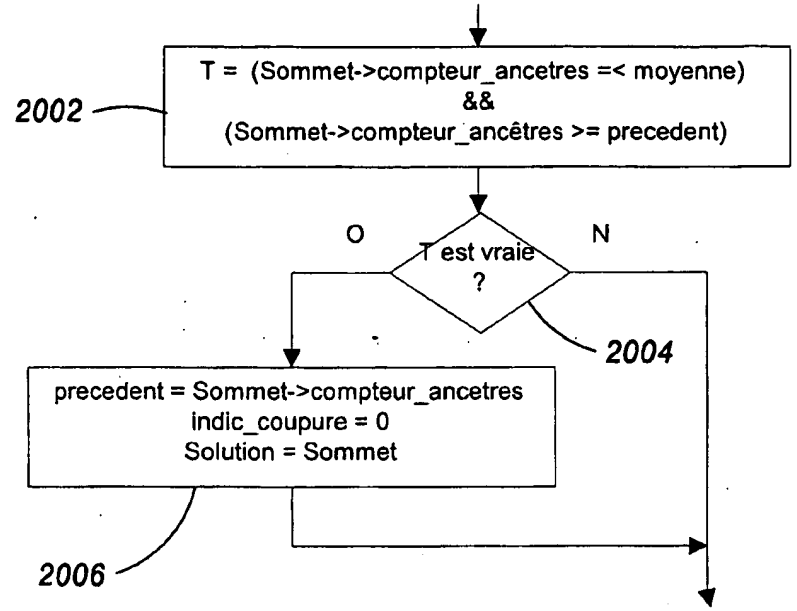

T = (Sommet->compteur_ancetres =< moyenne)
&&
(Sommet->compteur_ancêtres >= precedent)
2002
O
T est vraie ?
N
2004
precedent = Sommet->compteur_ancetres
indic_coupure = 0
Solution = Sommet
2006

FIG.20

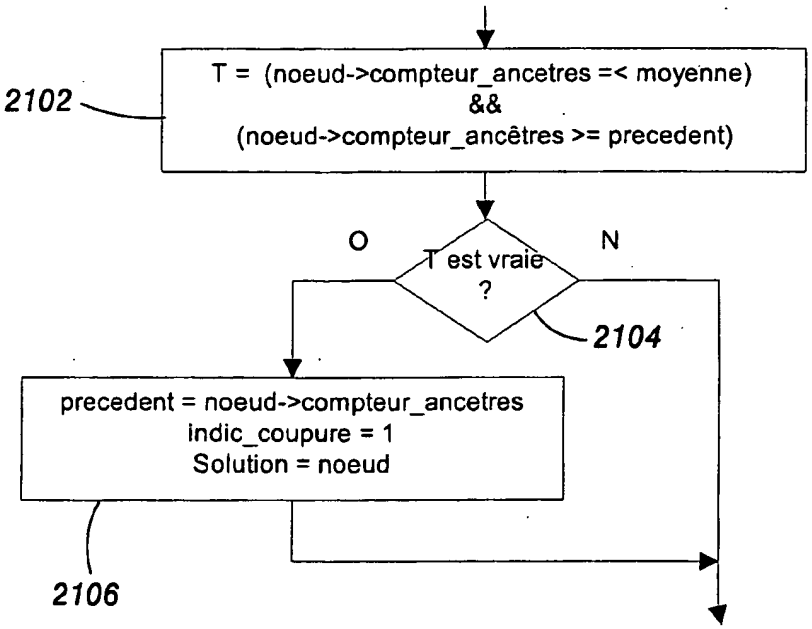

T = (noeud->compteur_ancetres =< moyenne)
&&
(noeud->compteur_ancêtres >= precedent)
2102
O
T est vraie ?
N
2104
precedent = noeud->compteur_ancetres
indic_coupure = 1
Solution = noeud
2106

FIG.21

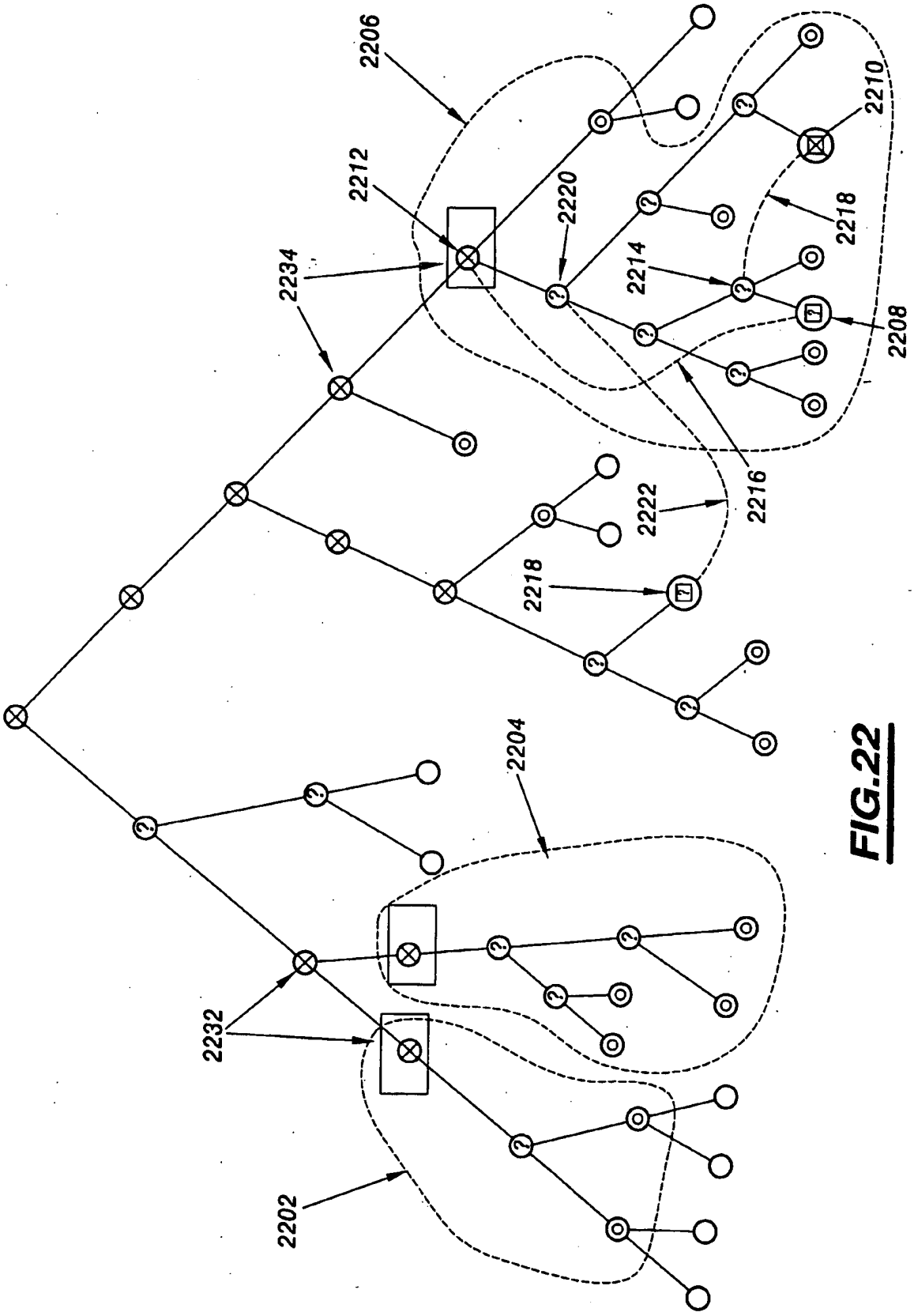

2206
2210
2212
2220
2218
2234
2214
2208
2216
2222
2218
2204
2232
2202

FIG.22

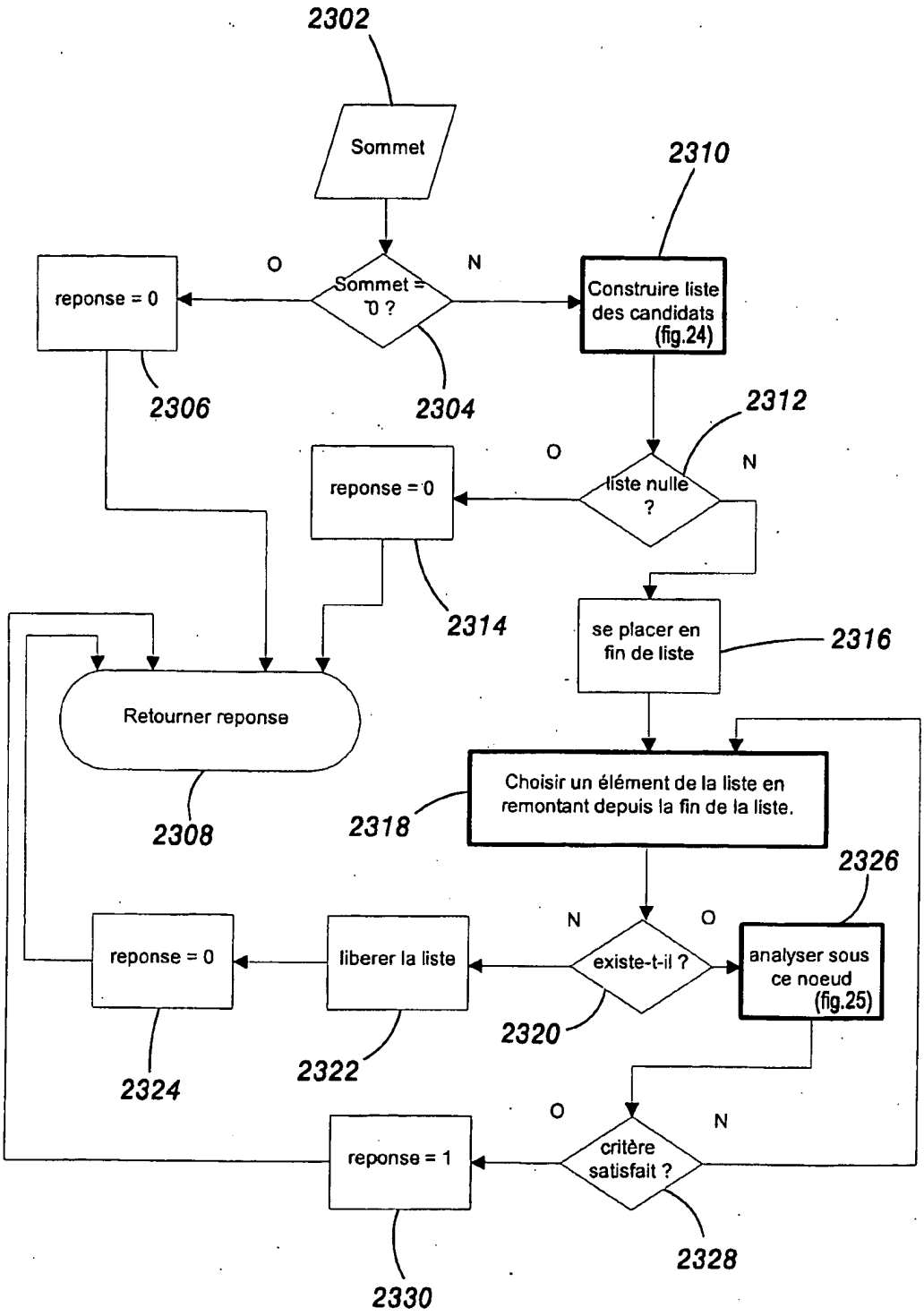

2302
Sommet
2310
O
N
reponse = 0
Sommet = 0 ?
Construire liste des candidats (fig.24)
2306
2304
2312
O
N
reponse = 0
liste nulle ?
2314
se placer en fin de liste
2316
Retourner reponse
2308
Choisir un élément de la liste en remontant depuis la fin de la liste.
2318
2326
N
O
reponse = 0
liberer la liste
existe-t-il ?
analyser sous ce noeud (fig.25)
2320
2322
2324
O
N
reponse = 1
critère satisfait ?
2328
2330

FIG.23

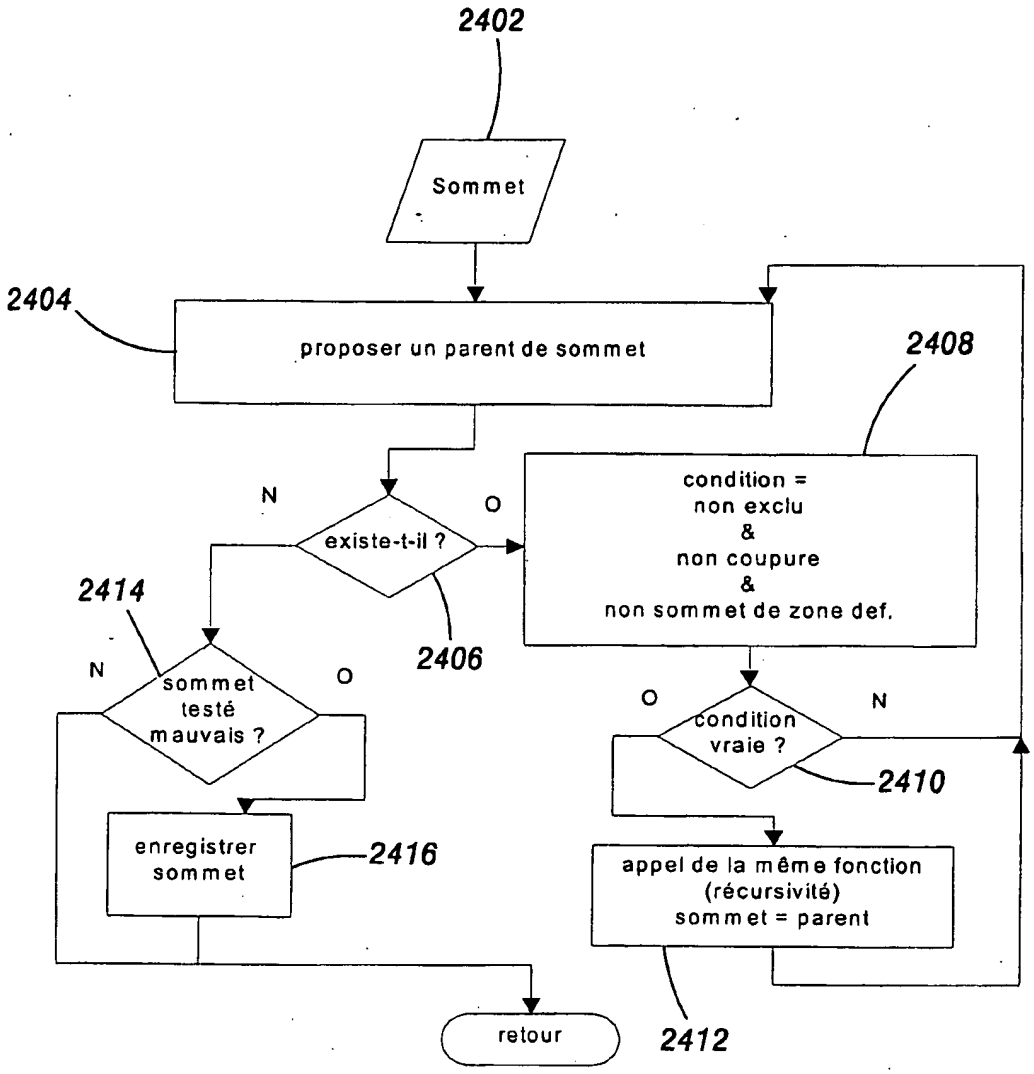

2402
Sommet
2404
proposer un parent de sommet
2408
condition =
non exclu
&
non coupure
&
non sommet de zone def.
N
O
existe-t-il ?
2406
2414
N
O
sommet
testé
mauvais ?
O
N
condition
vraie ?
2410
enregistrer
sommet
2416
appel de la même fonction
(récursivité)
sommet = parent
retour
2412

FIG.24

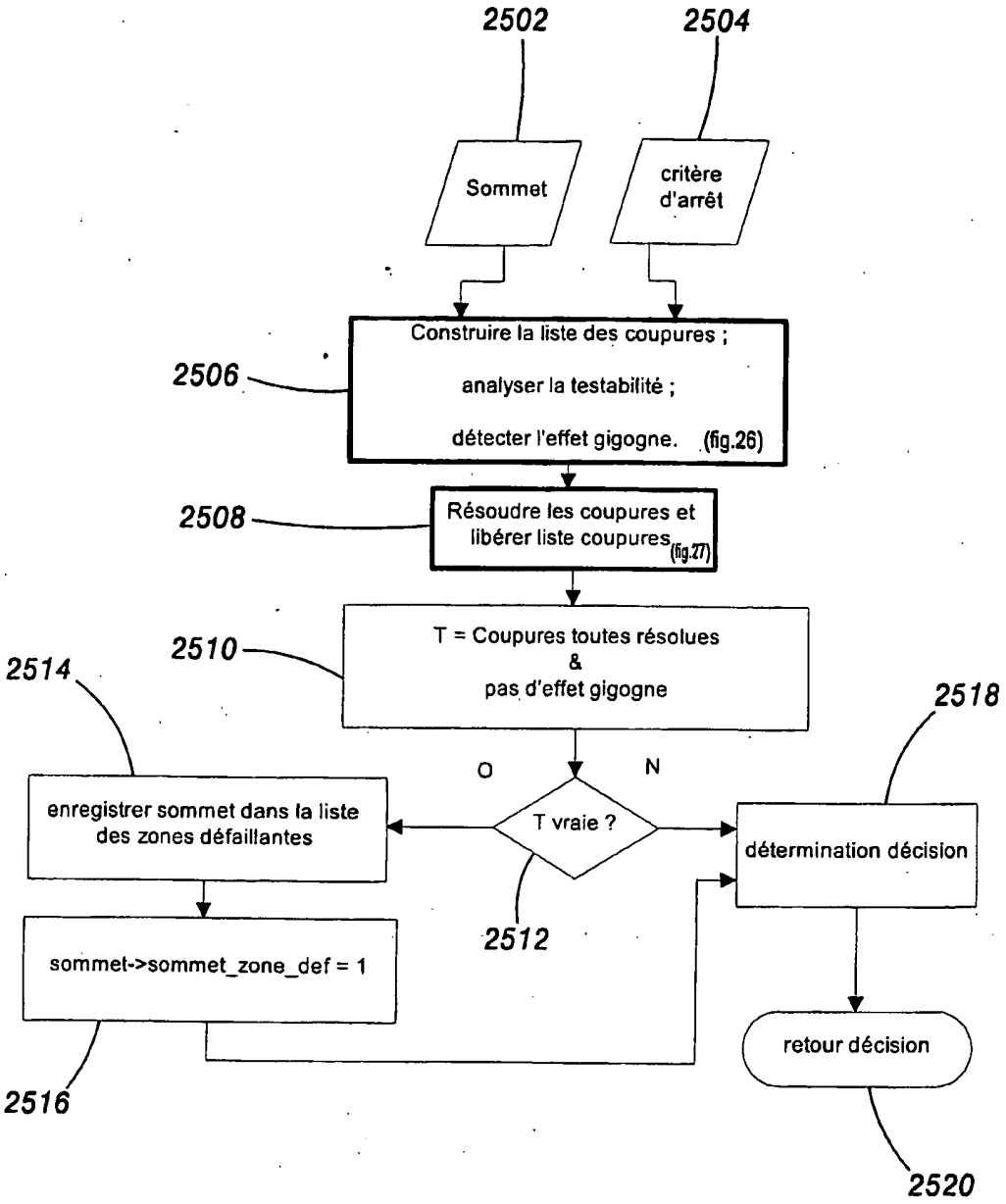

2502
2504
Sommet
critère d'arrêt
Construire la liste des coupures ;
analyser la testabilité ;
détecter l'effet gigogne. (fig.26)
2506
2508
Résoudre les coupures et libérer liste coupures (fig.27)
2510
T = Coupures toutes résolues & pas d'effet gigogne
2514
2518
O
N
enregistrer sommet dans la liste des zones défaillantes
T vraie ?
détermination décision
2512
sommet->sommet_zone_def = 1
retour décision
2516
2520

FIG.25

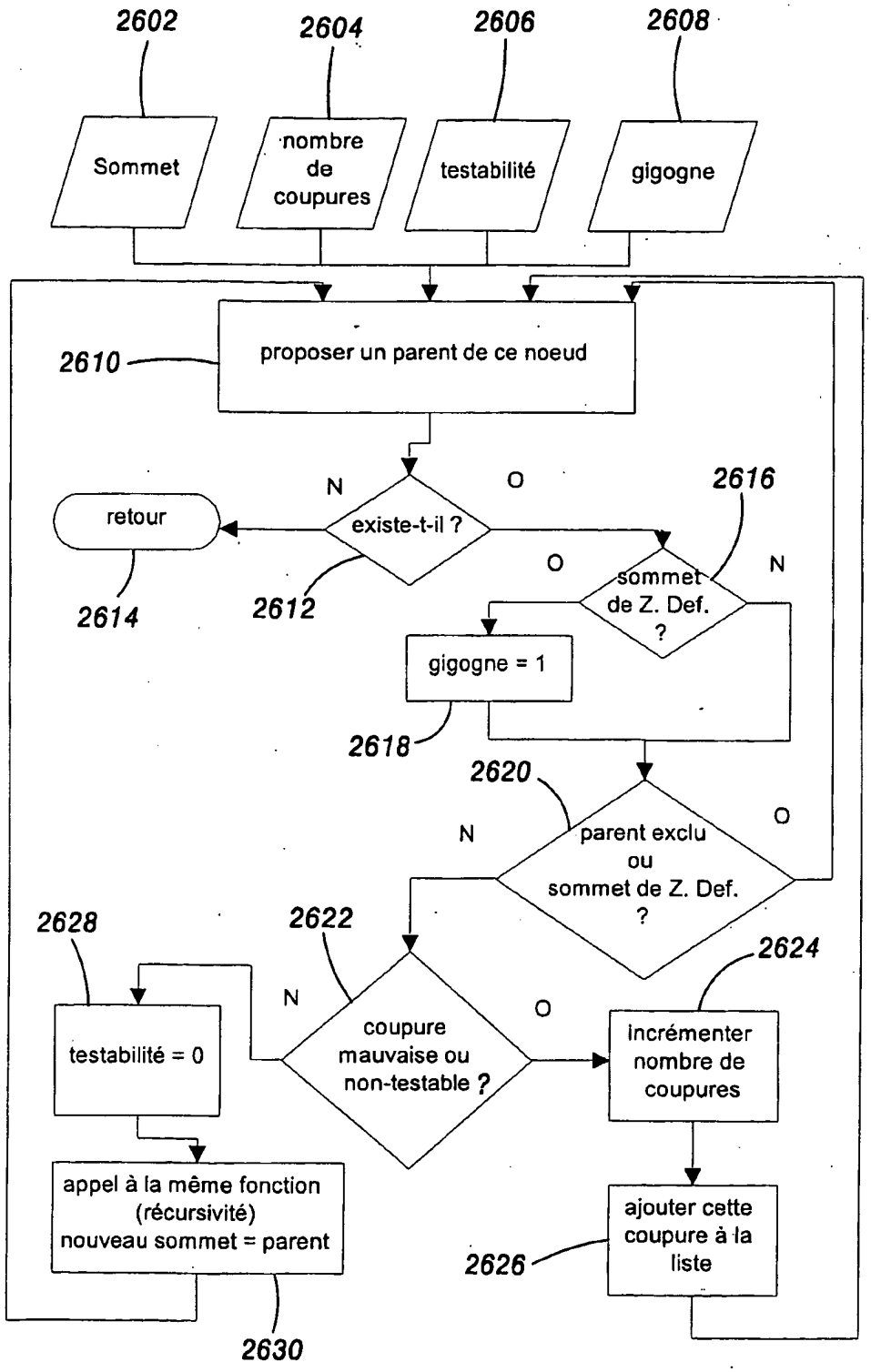

2602
2604
2606
2608
Sommet
nombre de coupures
testabilité
gigogne
2610
proposer un parent de ce noeud
N
O
2616
retour
existe-t-il ?
O
N
sommet de Z. Def. ?
2614
2612
gigogne = 1
2618
2620
N
O
parent exclu ou sommet de Z. Def. ?
2628
2622
2624
N
O
testabilité = 0
coupure mauvaise ou non-testable ?
incrémenter nombre de coupures
appel à la même fonction (récursivité) nouveau sommet = parent
ajouter cette coupure à la liste
2626
2630

FIG.26

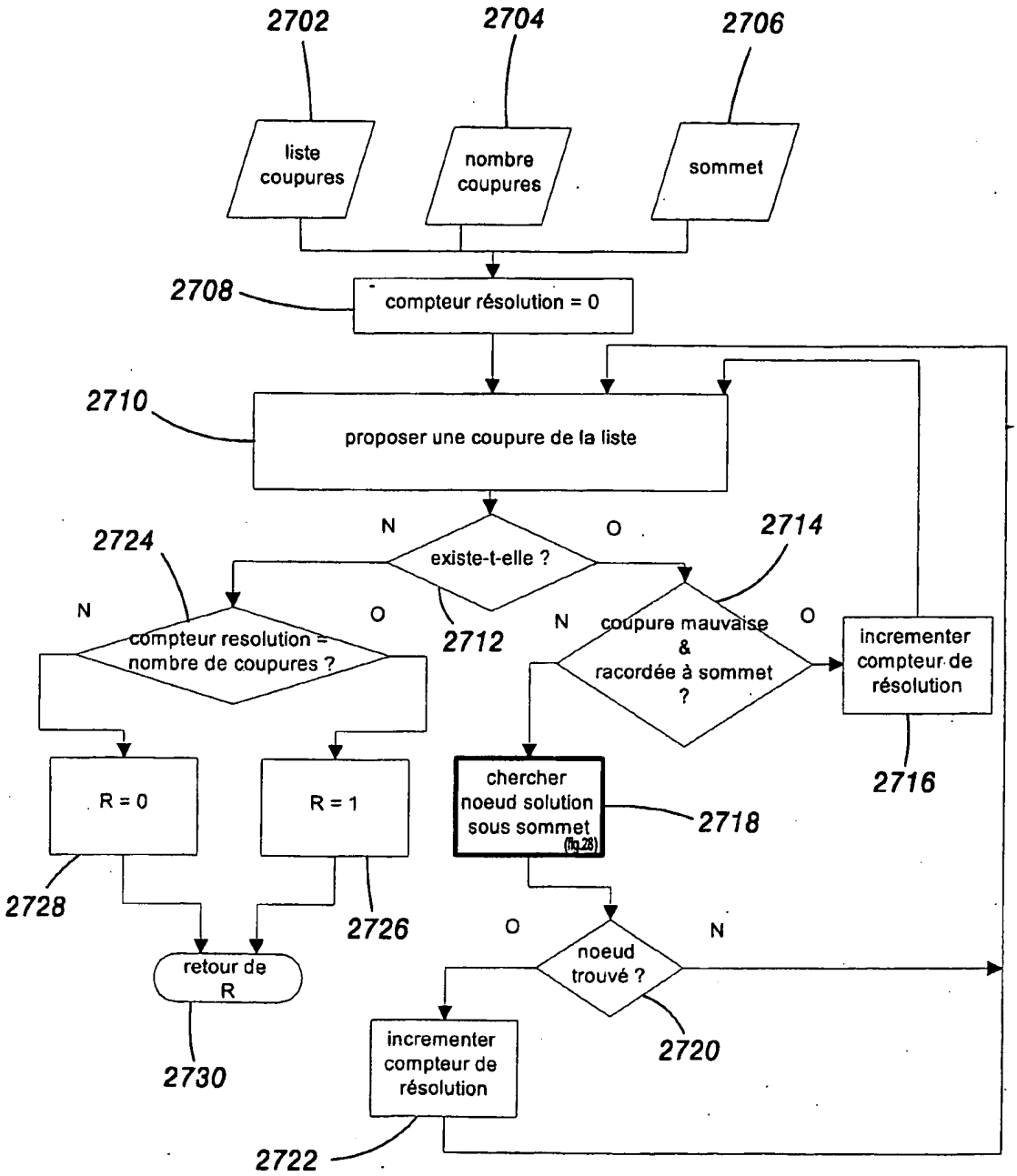

2702
2704
2706
liste coupures
nombre coupures
sommet
2708
compteur résolution = 0
2710
proposer une coupure de la liste
N
existe-t-elle ?
O
2714
2724
2712
N
compteur resolution = nombre de coupures ?
O
N
coupure mauvaise & racordée à sommet ?
O
incrementer compteur de résolution
2716
R = 0
R = 1
chercher noeud solution sous sommet
2718
2728
2726
O
noeud trouvé ?
N
retour de R
2720
incrementer compteur de résolution
2730
2722

FIG.27

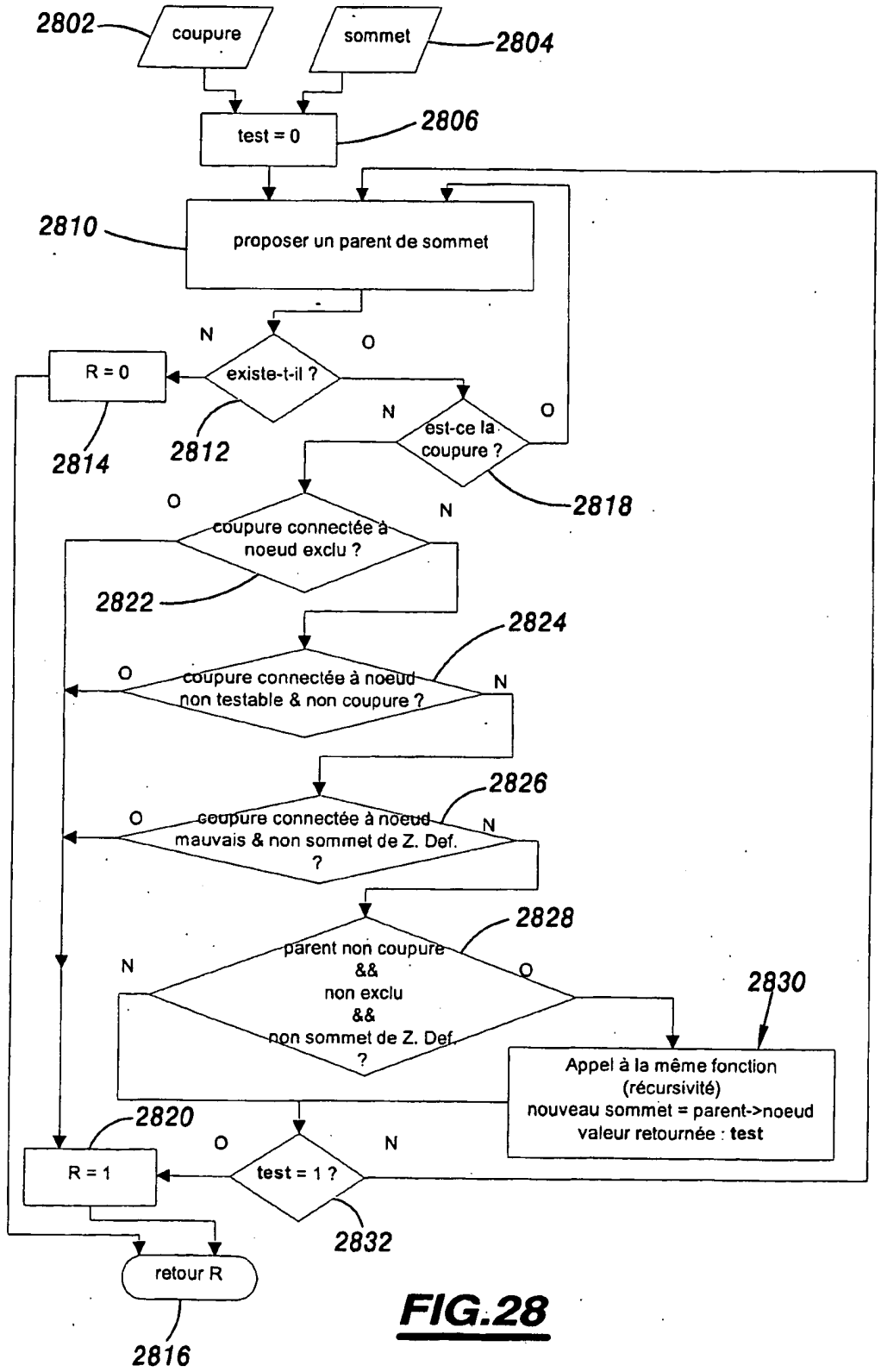

2802
coupure
sommet
2804
test = 0
2806
2810
proposer un parent de sommet
N
O
existe-t-il ?
R = 0
2814
2812
N
O
est-ce la coupure ?
2818
O
N
coupure connectée à noeud exclu ?
2822
2824
O
coupure connectée à noeud non testable & non coupure ?
N
2826
O
coupure connectée à noeud mauvais & non sommet de Z. Def. ?
N
2828
N
parent non coupure && non exclu && non sommet de Z. Def. ?
O
2830
Appel à la même fonction (récursivité)
nouveau sommet = parent->noeud
valeur retournée : test
2820
R = 1
O
test = 1 ?
N
2832
retour R
2816

FIG.28

INTERNATIONAL SEARCH REPORT

International Application No	PCT/FR 00/02554

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G01R31/3183

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G01R

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 305 217 A (HEWLETT PACKARD CO) 1 March 1989 (1989-03-01) abstract; claims 1-9	1-8
A	US 5 602 856 A (TERAMOTO MITSUO) 11 February 1997 (1997-02-11) abstract; claims 1,2; figures 4,5	1-8
A	EP 0 720 097 A (AT & T CORP) 3 July 1996 (1996-07-03) abstract; claims 1-4	1-8

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
8 February 2001	15/02/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Sarasua, L.

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/FR 00/02554

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0305217 A	01-03-1989	US 4853928 A	01-08-1989
		DE 3854636 D	07-12-1995
		DE 3854636 T	02-05-1996
		JP 1152379 A	14-06-1989
		KR 9206969 B	22-08-1992
US 5602856 A	11-02-1997	JP 6342038 A	13-12-1994
EP 0720097 A	03-07-1996	CA 2159036 A	30-06-1996
		JP 8304517 A	22-11-1996
		US 5559811 A	24-09-1996

Form PCT/ISA/210 (patent family annex) (July 1992)

RAPPORT DE RECHERCHE INTERNATIONALE

Demande internationale No
PCT/FR 00/02554

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
CIB 7 G01R31/3183

Selon la classification internationale des brevets (CIB) ou à la fois selon la classification nationale et la CIB

B. DOMAINES SUR LESQUELS LA RECHERCHE A PORTE

Documentation minimale consultée (système de classification suivi des symboles de classement)
CIB 7 G01R

Documentation consultée autre que la documentation minimale dans la mesure où ces documents relèvent des domaines sur lesquels a porté la recherche

Base de données électronique consultée au cours de la recherche internationale (nom de la base de données, et si réalisable, termes de recherche utilisés)
EPO-Internal

C. DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie.°	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	EP 0 305 217 A (HEWLETT PACKARD CO) 1 mars 1989 (1989-03-01) abrégé; revendications 1-9	1-8
A	US 5 602 856 A (TERAMOTO MITSUO) 11 février 1997 (1997-02-11) abrégé; revendications 1,2; figures 4,5	1-8
A	EP 0 720 097 A (AT & T CORP) 3 juillet 1996 (1996-07-03) abrégé; revendications 1-4	1-8

☐ Voir la suite du cadre C pour la fin de la liste des documents

☒ Les documents de familles de brevets sont indiqués en annexe

° Catégories spéciales de documents cités:

"A" document définissant l'état général de la technique, non considéré comme particulièrement pertinent

"E" document antérieur, mais publié à la date de dépôt international ou après cette date

"L" document pouvant jeter un doute sur une revendication de priorité ou cité pour déterminer la date de publication d'une autre citation ou pour une raison spéciale (telle qu'indiquée)

"O" document se référant à une divulgation orale, à un usage, à une exposition ou tous autres moyens

"P" document publié avant la date de dépôt international, mais postérieurement à la date de priorité revendiquée

"T" document ultérieur publié après la date de dépôt international ou la date de priorité et n'appartenenant pas à l'état de la technique pertinent, mais cité pour comprendre le principe ou la théorie constituant la base de l'invention

"X" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme impliquant une activité inventive lorsque le document est associé à un ou plusieurs autres documents de même nature, cette combinaison étant évidente pour une personne du métier

"&" document qui fait partie de la même famille de brevets

Date à laquelle la recherche internationale a été effectivement achevée	Date d'expédition du présent rapport de recherche internationale
8 février 2001	15/02/2001

Nom et adresse postale de l'administration chargée de la recherche internationale	Fonctionnaire autorisé
Office Européen des Brevets, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Sarasua, L.

1

Formulaire PCT/ISA/210 (deuxième feuille) (juillet 1992)

RAPPORT DE RECHERCHE INTERNATIONALE

Renseignements relatifs membres de familles de brevets

D le internationale No
FR/FR 00/02554

Document brevet cité au rapport de recherche		Date de publication	Membre(s) de la famille de brevet(s)		Date de publication
EP 0305217	A	01-03-1989	US	4853928 A	01-08-1989
			DE	3854636 D	07-12-1995
			DE	3854636 T	02-05-1996
			JP	1152379 A	14-06-1989
			KR	9206969 B	22-08-1992
US 5602856	A	11-02-1997	JP	6342038 A	13-12-1994
EP 0720097	A	03-07-1996	CA	2159036 A	30-06-1996
			JP	8304517 A	22-11-1996
			US	5559811 A	24-09-1996